Locosto BSP
Application Programming Interface

Technical Document

Document Number:	13_04_04_03073
TI Department	Cellular Systems Software Division
Version:	0.3
Status:	Draft
Date (mm-dd-yyyy):	16-MAY-2006

Important Notice

IMPORTANT NOTICE

Texas Instruments Incorporated and / or its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products, software and services at any time and to discontinue any product, software or service without notice. Customers should obtain the latest relevant information during product design and before placing orders and should verify that such information is current and complete.

All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgement. TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI products, software and / or services. To minimise the risks associated with customer products and applications, customers should provide adequate design, testing and operating safeguards.

Any access to and / or use of TI software described in this document is subject to Customers entering into formal license agreements and payment of associated license fees. TI software may solely be used and / or copied subject to and strictly in accordance with all the terms of such license agreements.

Customer acknowledges and agrees that TI products and / or software may be based on or implement industry recognised standards and that certain third parties may claim intellectual property rights therein. The supply of products and / or the licensing of software do not convey a license from TI to any third party intellectual property rights and TI expressly disclaims liability for infringement of third party intellectual property rights.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products, software or services are used.

Information published by TI regarding third–party products, software or services does not constitute a license from TI to use such products, software or services or a warranty, endorsement thereof or statement regarding their availability. Use of such information, products, software or services may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

WARNING:

Recipient agrees to not knowingly export or re-export, directly or indirectly, any product or technical data (as defined by the U.S, EU and other Export Administration Regulations) including software, or any controlled product restricted by other applicable national regulations, received from Disclosing party under this Agreement, or any direct product of such technology, to any destination to which such export or re-export is restricted or prohibited by U.S or other applicable laws, without obtaining prior authorisation from U.S. Department of Commerce and other competent Government authorities to the extent required by those laws. This provision shall survive termination or expiration of this Agreement. According to our best knowledge of the state and end-use of this product or technology, and in compliance with the export control regulations of dual-use goods in force in the origin and exporting countries, this technology is classified as follows:

-US ECCN: 5E991
-EU ECCN: 5E991

and may require export or re-export license for shipping it in compliance with the applicable regulations of certain countries.

Topics

Chapter 1 CAMD DRIVER

1.1 Introduction

This document describes the GPF CAMD API for the Locosto camera module driver.
The API offers straightforward access to the camera module. To this end, the CAMD ENTITY uses services offered by the lower level drivers to provide

- camera sensor control (e.g. image size, gamma correction)
- data acquisition (viewfinder frames, snapshots)

All API entries honour the Return Path concept, which means that the client can be notified of the result of a service request by

- Receiving a message in its mailbox *or*
- Using a callback function.

When using a callback function, the response to the service request is passed as a parameter to this function.

If a message is received from the CAMD driver, the client is responsible for releasing any associated memory.

1.2 Interface description Camera Driver

In this paragraph the interface of the camera driver is described.

1.3 Driver functions definition

1.3.1 camd_registerclient

```
T_RV_RET camd_registerclient (BOOL enable_sensor, T_RV_RETURN rp)
```

Description

This function switches the sensor between the enabled and disabled state. In the disabled state the sensor might use less power. It is disabled by default. When switching to the enabled state, the driver will perform the necessary hardware initialisations. The camera must be enabled before any other API entries (except `camd_get_sw_version()`) may be used.

Parameters

- **`enable_sensor`**

TRUE enables the sensor, FALSE disables the sensor.

- **`rp`**

Return path.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_NOT_READY	CAMD SWE is not running
RV_MEMORY_ERR	Not enough memory

Event Return

A CAMD_STATUS_RSP_MSG indicating success or failure is returned.

Current restriction of use

None.

1.3.2 camd_set_configparams

```
T_RV_RET camd_set_configparams (T_CAMD_PARAMETERS * param_p,
                                T_RV_RETURN rp)
```

Description

This function sets all parameters.

Parameters

- **param_p**

Pointer to a parameter blocks containing the parameters to be used. See section **Error! Reference source not found.** for a description of parameters.

- **rp**

Return path.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_NOT_READY	CAMD SWE is not running or sensor not enabled
RV_MEMORY_ERR	Not enough memory

Event Return

A CAMD_STATUS_RSP_MSG indicating success or failure is returned.

Current restriction of use

The sensor must be enabled.

1.3.3 camd_get_configparams

```
T_RV_RET camd_get_configparams (T_RV_RETURN rp)
```

Description

This function fetches all parameters currently in use by the driver.

Parameters

- **rp**

Return path.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_NOT_READY	CAMD SWE is not running or sensor not enabled
RV_MEMORY_ERR	Not enough memory

Event Return

A `CAMD_GET_CONFIGPARAMS_RSP_MSG` is returned. See section **Error! Reference source not found.** for a description of viewfinder or snapshot parameters.

Current restriction of use

The sensor must be enabled.

1.3.4 camd_usebuff

```
T_RV_RET camd_usebuff (UINT8 *buff, T_RV_RETURN rp)
```

Description

This function captures a snapshot image and stores it.

Parameters

- **rp**

The return path.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_NOT_READY	CAMD SWE is not running or sensor not enabled
RV_MEMORY_ERR	Not enough memory

Event Return

A `CAMD_SNAPSHOT_DATA_RSP_MSG or CAMD_VIEWFINDER_DATA_RSP_MSG` is returned by CAMD after the snapshot or viewfinder data has been captured and stored.

Current restriction of use

The sensor must be enabled. Destination memory and data format and mode must be selected first with `camd_set_configparams()`.

1.3.5 camd_get_sw_version

```
UINT32 camd_get_sw_version(void)
```

Description

This function returns the driver version.

Parameters

None.

Immediate Return

- **UINT32**

Bit	Name	Function
[0-15]	BUILD	Build number
[16-23]	MINOR	Minor version number
[24-31]	MAJOR	Major version number

Event Return

None.

Current restriction of use

None.

1.4 Message definition

Some of the message fields are common to all requests and responses. These fields are described in the next two sections, which assume the definition of a `T_CAMD_MSG *msg_p`.

1.4.1 Request

Generally, a CAMD request is formed as follows:

```
msg_p->os_hdr.msg_id = CAMD_xxx_REQ_MSG
msg_p->rp = <return path>
```

Other fields may be defined depending on the message ID. Note that it is more convenient to use the function call API to do requests.

1.4.2 Response

The CAMD driver will respond to a request with the following message:

```
msg_p->os_hdr.msg_id = CAMD_xxx_RSP_MSG
msg_p->status = <CAMD_OK | CAMD_INVALID_PARAMETER | CAMD_NOT_READY |
                 CAMD_TRANSFER_COMPLETE>
```

Other fields may be defined depending on the message ID.

Status values indicating normal operation	
`CAMD_OK`	request is being processed
`CAMD_TRANSFER_COMPLETE`	data transfer (snapshot or viewfinder frame) complete

If the status is not `CAMD_OK` or `CAMD_TRANSFER_COMPLETE`, all other fields of the response are undefined. Some API calls result in more than one response message. If a response to such a call indicates an error, no further responses will be sent.

Status values indicating errors	
CAMD_INVALID_PARAMETER	indicates that the request contained one or more invalid parameters, e.g. a range error an unexpected NULL pointer
CAMD_NOT_READY	indicates that the driver is not ready to handle the request, because the CAMD task is either not running (and initialized) or because the sensor is not enabled

1.4.3 CAMD_STATUS_RSP_MSG

This message is returned to indicate the successful completion or failure of the following requests:
- Setting parameters, and
- Acquisition of frame data.

The following fields are defined:

```
msg_p->os_hdr.msg_id = CAMD_STATUS_RSP_MSG
msg_p->status = <CAMD_OK | CAMD_INVALID_PARAMETER | CAMD_NOT_READY |
                 CAMD_TRANSFER_COMPLETE>
```

See section **Error! Reference source not found.** for a description of the status field.

1.4.4 CAMD_REGISTERCLIENT_REQ_MSG

This message enables or disables the sensor.

The following fields are defined:

```
msg_p->os_hdr.msg_id = CAMD_REGISTERCLIENT_REQ_MSG
msg_p->rp = <return path>
msg_p->body.enable_sensor       /* TRUE = enable, FALSE = disable */
```

A CAMD_STATUS_RSP_MSG is returned by CAMD.

1.4.5 CAMD_SET_CONFIGPARAMS_REQ_MSG

This message sends a block of parameters to the CAMD SWE.

The following fields are defined:

```
msg_p->os_hdr.msg_id = CAMD_SET_CONFIGPARAMS_REQ_MSG
msg_p->rp = <return path>
msg_p->body.configparams.capturemode
msg_p->body.configparams.resolution
msg_p->body.configparams.mode
msg_p->body.configparams.encoding
msg_p->body.configparams.gamma_correction
msg_p->body.configparams.imagewidth
msg_p->body.configparams.imageheight
msg_p->body.configparams.black_and_white
msg_p->body.configparams.flip_x
msg_p->body.configparams.flip_y
msg_p->body.configparams.rotate
msg_p->body.configparams.zoom
```

See section **Error! Reference source not found.** for a description of the parameters. A CAMD_STATUS_RSP_MSG is returned by CAMD.

1.4.6 CAMD_GET_CONFIGPARAMS_REQ_MSG

This message requests the snapshot parameters currently in use by the CAMD SWE.

The following fields are defined:

```
msg_p->os_hdr.msg_id = CAMD_GET_SNAPSHOT_PARAMETERS_REQ_MSG
msg_p->rp = <return path>
```

See section 6.1.5 for a description of the snapshot parameters.
A `CAMD_GET_SNAPSHOT_PARAMETERS_RSP_MSG` is returned by CAMD.

1.4.7 CAMD_GET_CONFIGPARAMS_RSP_MSG

This message returns the parameters currently in use by the CAMD SWE.

The following fields are defined:

```
msg_p->os_hdr.msg_id = CAMD_GET_CONFIGPARAMS_RSP_MSG
msg_p->body.configparams.capturemode
msg_p->body.configparams.resolution
msg_p->body.configparams.mode
msg_p->body.configparams.encoding
msg_p->body.configparams.gamma_correction
msg_p->body.configparams.imagewidth
msg_p->body.configparams.imageheight
msg_p->body.configparams.black_and_white
msg_p->body.configparams.flip_x
msg_p->body.configparams.flip_y
msg_p->body.configparams.rotate
msg_p->body.configparams.zoom
```

See section **Error! Reference source not found.** for a description of the parameters. A `CAMD_STATUS_RSP_MSG` is returned by CAMD.

1.4.8 CAMD_USEBUFF_REQ_MSG

This message triggers the CAMD to take a snapshot or a viewfinder image and store the result in the preselected memory bank, in the preselected format.

The following fields are defined:

```
msg_p->os_hdr.msg_id = CAMD_USEBUFF_REQ_MSG
msg_p->rp = <return path>
```

A `T_CAMD_SNAPSHOT_DATA_RSP_MSG` or `CAMD_VIEWFINDER_DATA_RSP_MSG` indicating `CAMD_TRANSFER_COMPLETE` is returned by CAMD.

1.4.9 CAMD_VIEWFINDER_DATA_RSP_MSG

This message is sent by CAMD to indicate the arrival of new viewfinder image.

The following fields are defined:

```
msg_p->os_hdr.msg_id = CAMD_VIEWFINDER_DATA_RSP_MSG
msg_p->status = < CAMD_TRANSFER_COMPLETE | CAMD_MEMORY_ERROR |
                  CAMD_NOT_READY | CAMD_INTERNAL_ERR >
```

1.4.10 CAMD_SNAPSHOT_DATA_RSP_MSG

This message is sent by CAMD to indicate the arrival of new snapshot image.

The following fields are defined:

```
msg_p->os_hdr.msg_id = CAMD_SNAPSHOT_DATA_RSP_MSG
msg_p->status = < CAMD_TRANSFER_COMPLETE | CAMD_INVALID_PARAMETER |
                CAMD_NOT_READY | CAMD_INTERNAL_ERR >
```

1.5 Type definition

1.5.1 T_CAMD_RESOLUTION

```
typedef enum { CAMD_VGA, CAMD_QCIF } T_CAMD_RESOLUTION;
```

1.5.1 T_CAMD_ENCODING

```
typedef enum { CAMD_YUYV_INTERLEAVED, CAMD_RGB_888, CAMD_RGB_565 }
T_CAMD_ENCODING
```

Note: for viewfinder frames, only `T_CAMD_RGB_888 is not supported for any mode.`

1.5.2 T_CAMD_VIEWFINDER_MODE

```
typedef enum { CAMD_SINGLE_SHOT, CAMD_CONTINUOUS } T_CAMD_VIEWFINDER_MODE
```

Note: only `T_CAMD_CONTINUOUS` is supported.

1.5.3 T_CAMD_GAMMA

```
typedef enum  { CAMD_GAMMA_CORR_DEFAULT, CAMD_GAMMA_CORR_3_2 } T_CAMD_GAMMA;
```

Specifies the gamma correction to be used by the sensor. The valid range is from `CAMD_GAMMA_MIN` to `CAMD_GAMMA_MAX`. `CAMD_GAMMA_NEUTRAL` means no gamma correction.

1.5.4 T_CAMD_ PARAMETERS

```
typedef struct
  {
       BOOL capturemode;
       T_CAMD_RESOLUTION resolution;
       T_CAMD_VIEWFINDER_MODE mode;
       T_CAMD_ENCODING encoding;
       T_CAMD_GAMMA gamma_correction;
       UINT16 imagewidth;
       UINT16 imageheight;
       BOOL black_and_white;
       BOOL flip_x;
       BOOL flip_y;
       UINT16 rotate;
       UINT16 zoom;
       void (*start_transfer_cb) (void);
  }
  T_CAMD_PARAMETERS;
```

This type specifies the snapshot parameters to be used by CAMD:

- `Capturemode: viewfinder or snapshot mode`
- `mode: viewfinder operation mode`
- `resolution` : image size

- `encoding`: how to encode the pixel data
- `gamma_correction`: gamma correction value
- `imagewidth`: image width
- `imageheight`: image height
- `black_and_white`: black and white mode
- `flip_x`: flip x direction
- `flip_y`: flip y direction
- `rotate`: rotation: 0 = no rotation, 1 = 90 degrees counterclockwise, etc.
- `zoom`: zoom level, 0 = no zoom, 1 = 2x zoom, etc.

1.5.5 T_CAMD_MSG

The framework for all CAMD messages is defined as follows:

```
typedef struct
  {
  T_RV_HDR os_hdr;
  T_RV_RET status;
  T_RV_RETURN rp;
  union
  {
    T_CAMD_PARAMETERS configparams;
    BOOL enable_sensor;
    UINT8 *buff;
  }
  body;
  }
  T_CAMD_MSG;
```

1.6 ENTITY State diagram

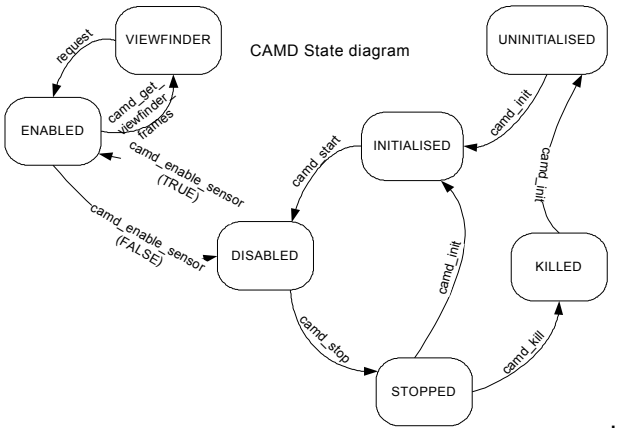


Figure 1 ENTITY diagram for CAMD

Figure 1 State Diagram

States		
Un-initialized		
Description: In this state the SWE is not initialised and not active.		
Accepted Messages/Functions: None.		
Event	**to state**	**Description**
camd_init()	initialised	Changing the state from “un-initialised” to “initialised” requires the following events: The Operating system needs to call the generic functions camd_get_info() and camd_set_info() then the camd_init() shall be called.
Initialised		
Description: In this state the SWE is initialised. The driver functionality is not available at this point.		
Accepted Messages/Functions: None.		
Event	**to state**	Description
camd_start()	Idle	The Operating system shall call the camd_start() after which the entity enters the “idle” state.
Disabled		
Description: In this state the SWE is ready to handle requests from the client. The sensor HW is disabled.		
Accepted Messages/Functions: camd_get_sw_version(), camd_registerclient()		
Event	**to state**	Description
CAMD_REGISTERCLIENT_REQ_MSG(true)	Enabled	The sensor HW is enabled.

camd_stop()	Stopped	When this function is called the driver will go into stopped state.

Enabled

Description:
In this state the SWE is ready to handle requests from the client. The sensor HW is enabled.

Accepted Messages/Functions:

Event	**to state**	Description
CAMD_REGISTERCLIENT_REQ_MSG (false)	disabled	The sensor HW is disabled.
CAMD_SET_CONFIGPARAMS()	viewfinder/ snapshot	configure sensor for one of the mode.
CAMD_USEBUFF()	viewfinder/ snapshot	Acquire data frame.

Viewfinder

Description:
In this state the sensor is ready to generate viewfinder frames.

Accepted Messages/Functions:
All CAMD functions.

Event	**to state**	Description
any request	enabled	Any function will return CAMD to the enabled state.

Stopped

Description:
In this state the SWE will be stopped and not ready for any requests

Accepted Messages/Functions:
None.

Event	**to state**	Description
camd_init()	initialised	Changing the state from “un-initialised” to “initialised” requires the following events: The Operating system needs to call the generic functions camd_get_info() and camd_set_info() then the camd_init() shall be called.
camd_kill()	killed	Changing from Stopped to Killed requires that the Operating system calls camd_kill(). In this function the xxx swe will free its allocated memory.

Killed

Description:
In this state the driver is inactive and will not be ready for any requests

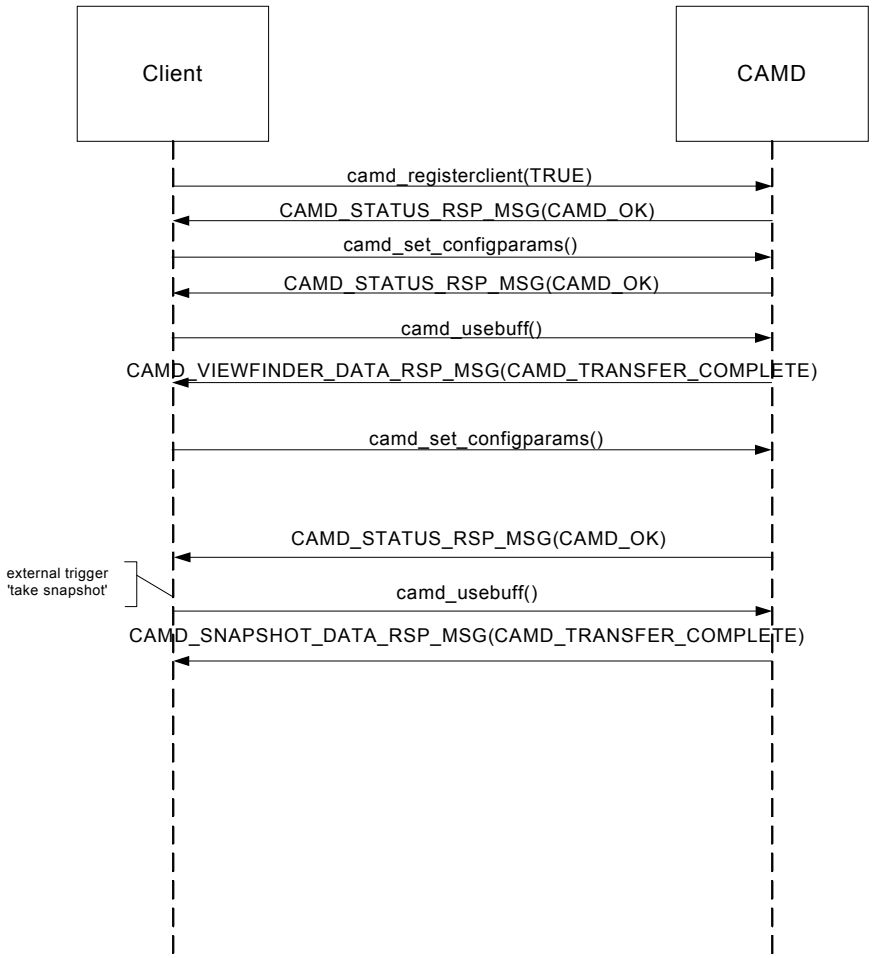
Accepted Messages/Functions:
None.

Event	**to state**	**Description**
camd_init()	initialised	Changing the state from “un-initialised” to “initialised” requires the following events: The Operating system needs to call the generic functions camd_get_info() and camd_set_info() then the camd_init() shall be called.

1.7 Usage Scenarios

1.7.1 Reading viewfinder frames and taking a snapshot

This section shows how to read viewfinder frames, until some external trigger calls for a snapshot. A typical application would restart the viewfinder mode with `camd_get_viewfinder_frames ()` after the snapshot is complete.



Chapter 2 CAMCORE

2.1 Introduction

This document describes the API of Camera Core. Camera Core module can interface with variety of external image sensors. It stores the image data in a FIFO and can generate DMA request.

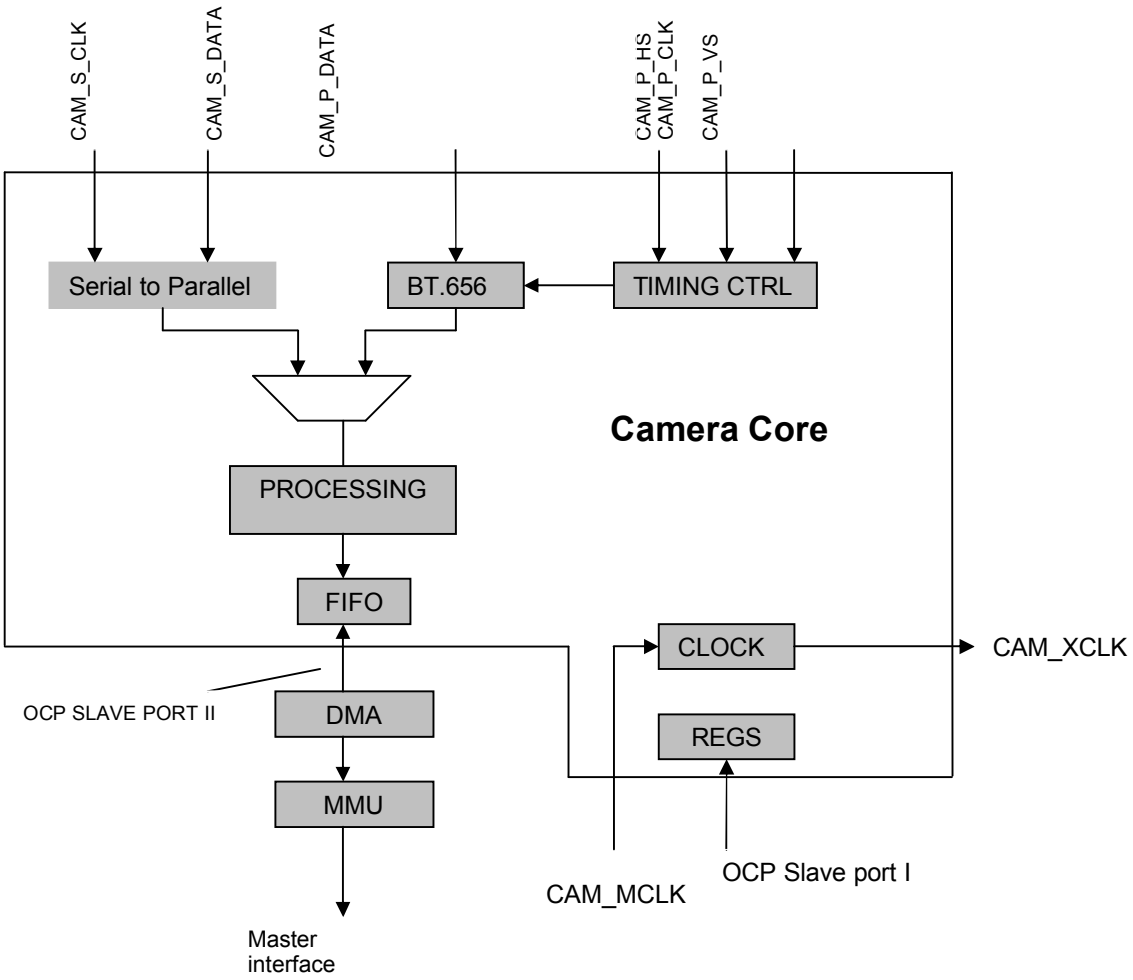


Figure 2 the camera core module interface

The above figure shows the Camera Core top level diagram and how the camera core module can be connected to the rest of the system.

2.2 Interface description Application

In the following sections the APIs of Camera Core are described.

2.2.1 camcore_config

```
T_CAMCORE_RET camcore_config(T_CAMCORE_CONFIGPARAM *params)
```

Description

This API would configure the camera core based on the desired input parameters. The input parameter would typically depend on: external sensor, chip-clock requirements etc. Camera needs to be enabled only after it is configured.

Parameters

- ***param**

It specifies the parameters necessary to configure the camera core. The parameters are

T_CAMCORE_MODE: States whether the camera is in viewfinder or snapshot mode.

T_CAMCORE_FIFOSIZE: Software is not allowed to change this value. FIFO size is fixed for camera core.

T_CAMCORE_THRESHOLD: size of FIFO threshold; software should configure this value for optimal data flow.

vsynch: VSYNCH is not available for Locosto. That's why it is always set to be ON.

T_CAMCORE_CCPMODE: Identifies the mode of the data flow. NoBT, BT656..etc..

xclk_div: CAM_XCLK = CAM_MCLK/ xclk_div. If this value is -1, then sensor is supplied with its own clock.

Immediate Return

- **T_CAMCORE_RET**

The possible values are:

id	Definition
CAMCORE_OK	The API function was successfully executed. Expect status message.
CAMCORE_INTERNAL_ERROR	Some internal error has occurred. Configuration is not Successful.
CAMCORE_NOT_SUPPORTED	Configuration is not Successful. Cam core is not ready to be configured.
CAMCORE_INVALID_PARAMETERS	Configuration is not Successful. The input parameters are not valid.

2.2.2 camcore_enable

```
T_CAMCORE_RET camcore_enable(void);
```

Description

The API would enable the camera core. This function needs to be called only after camera is configured.

Parameters

void

Immediate Return

- **T_CAMCORE_RET**

The possible values are:

id	Definition
CAMCORE_OK	The API function was successfully executed. Expect status message.
CAMCORE_INTERNAL_ERROR	Some internal error has occurred. Configuration is not Successful.
CAMCORE_NOT_SUPPORTED	Configuration is not Successful. Cam core is not ready to be configured.
CAMCORE_INVALID_PARAMETERS	Configuration is not Successful. The input parameters are not valid.

Current Restriction

This function needs to be called only after camera is configured.

2.2.3 camcore_disable

```
T_CAMCORE_RET camcore_disable(void);
```

Description

The API would disable the camera core.

Parameters

None.

Immediate Return

- **T_CAMCORE_RET**

The possible values are:

id	Definition
CAMCORE_OK	The API function was successfully executed. Expect status message.
CAMCORE_INTERNAL_ERROR	Some internal error has occurred. Configuration is not Successful.
CAMCORE_NOT_SUPPORTED	Configuration is not Successful. Cam core is not ready to be configured.
CAMCORE_INVALID_PARAMETERS	Configuration is not Successful. The input parameters are not valid.

Current Restriction

This function needs to be called only after camera is configured and enabled.

2.2.4 camcore_getRevision

```
T_CAMCORE_RET camcore_getRevision(T_CAMCORE_REVISION *revision);
```

Description

Reads the IP Revision code from CC_REVISION register. In the register [7:4] bits give Major revision and [3:0] bits give Minor revision.

Parameters

- ***revision**

Pointer to the T_CAMCORE_REVISON structure, which consists of field major and minor number.

Immediate Return

- **T_CAMCORE_RET**

The possible values are:

id	Definition
CAMCORE_OK	The API function was successfully executed. Expect status message.
CAMCORE_INTERNAL_ERROR	Some internal error has occurred. Configuration is not Successful.
CAMCORE_NOT_SUPPORTED	Configuration is not Successful. Cam core is not ready to be configured.
CAMCORE_INVALID_PARAMETERS	Configuration is not Successful. The input parameters are not valid.

Current restriction of use

None.

2.2.5 camcore_reset

```
T_CAMCORE_RET camcore_reset(T_CAMCORE_RESET_TYPE reset_type);
```

Description

The API would reset the camera core depending on the 'reset_type' required.

Parameters

- **reset_type**

This parameter passes the structure T_CAMCORE_RESET_TYPE to reset the Cam Core. The possible modes of reset are

CAMCORE_RESET_ALL: The API would reset the whole camera core; camcore_config needs to be called after this.

CAMCORE_RESET_FSM: Resets all the internal finite state machines of the camera core module. Must be applied when CC_EN = 0. Configuration settings will not be altered here.

Immediate Return

- **T_CAMCORE_RET**

The possible values are:

id	Definition
CAMCORE_OK	The API function was successfully executed. Expect status message.
CAMCORE_INTERNAL_ERROR	Some internal error has occurred. Configuration is not Successful.
CAMCORE_NOT_SUPPORTED	Configuration is not Successful. Cam core is not ready to be configured.
CAMCORE_INVALID_PARAMETERS	Configuration is not Successful. The input parameters are not valid.

Current restriction of use

None.

2.2.6 camcore_setGpioPins

```
void camcore_setGpioPins(void);
```

Description

Sets the General purpose IO pins for Camera Core.

CONF_GPIO_0	Cam_d3	0xF100	10
CONF_GPIO_19	Cam_hs	0xF16A	001
CONF_GPIO_20	Cam_d3	0xF16C	010
CONF_GPIO_21	Cam_lclk	0xF16E	01
CONF_GPIO_22	Cam_slck	0xF170	001
CONF_GPIO_28	D7	0xF17C	11
CONF_GPIO_29	D6	0xF17E	11
CONF_GPIO_30	D5	0xF180	11
CONF_ND_NWP	D4	0xF184	10
CONF_GPIO_47	D0	0xF1B6	10
CONF_GPIO_7	D2	0xF1BA	101

Parameters

- **void**

Immediate Return

Void

Current restriction of use

None.

2.2.7 camcore_setmode

```
T_CAMCORE_RET camcore_setmode(T_CAMCORE_MODE mode);
```

Description

The API would set the Cam Core either on View finder or in Snapshot mode, depending on the argument passes.

Parameters

- **mode**

Defines the different mode of settings of the Cam Core. Especially two modes are there, SNAPSHOT and VIEWFINDER.

Immediate Return

- **T_CAMCORE_RET**

The possible values are:

id	Definition
CAMCORE_OK	The API function was successfully executed. Expect status message.
CAMCORE_INTERNAL_ERROR	Some internal error has occurred. Configuration is not Successful.
CAMCORE_NOT_SUPPORTED	Configuration is not Successful. Cam core is not ready to be configured.
CAMCORE_INVALID_PARAMETERS	Configuration is not Successful. The input parameters are not valid.

Current restriction of use

None.

2.3 Type definitions and constants

API type definitions and constants are located in the configuration file img_api.h in the common directory.

2.3.1 T_CAMCORE_MODE

Specifies different modes of Camera operation.

```
typedef enum {
   CAMCORE_SNAPSHOT = 0,
   CAMCORE_VIEWFINDER
} T_CAMCORE_MODE;
```

2.3.2 T_CAMCORE_CCPMODE

Specifies different CCP (Compact Camera Port) mode, for example, CCP serial sensor interface or parallel interface etc.

```
typedef enum {
   CAMCORE_CCP_EN = 0,
   CAMCORE_CCP_PARNOBT_8,
   CAMCORE_CCP_PARNOBT_10,
   CAMCORE_CCP_PARNOBT_12,
   CAMCORE_CCP_RESV_1,
   CAMCORE_CCP_PARBT_8,
   CAMCORE_CCP_PARBT_10,
   CAMCORE_CCP_RESV_2,
   CAMCORE_CCP_FIFO_TEST

} T_CAMCORE_CCPMODE;
```

2.3.3 T_CAMCORE_RETURN

Defines all possible return types from Cam Core.

```
        typedef enum {
        CAMCORE_OK = 0,
        CAMCORE_INTERNAL_ERR,
        CAMCORE_NOT_SUPPORTED,
        CAMCORE_INVALID_PARAMS
} T_CAMCORE_RET;
```

2.3.4 T_CAMCORE_RESET_TYPE

Defines the possible options to reset Cam Core.

```
typedef enum {
                CAMCORE_RESET_ALL = 0,
                CAMCORE_RESET_FSM
        }T_CAMCORE_RESET_TYPE;
```

2.3.5 T_CAMCORE_FIFOSIZE

typedef UINT16 T_CAMCORE_FIFOSIZE;

Sets the Cam Core FIFO Size. The number of 32 bit words in the camera core FIFO is $2^{(FIFOSIZE)}$. FIFOSIZE can be any value of the range [1-7], in the following table.

Generic Parameter	Default Value	Description
T_CAMCORE_FIFOSIZE	6	1: 2*32 bits word data RAM
		2: 4*32 bits word data RAM
		3: 8*32 bits word data RAM
		4:16*32 bits word data RAM
		5:32*32 bits word data RAM
		6:64*32 bits word data RAM
		7:128*32 bits word data RAM

In present case FIFOSIZE is set as 128, the maximum.

2.3.6 T_CAMCORE_THRESHOLD

typedef UINT16 T_CAMCORE_THRESHOLD;

Sets a threshold for the FIFO. In present case it is set as 64.

2.3.7 T_CAMCORE_CONFIGPARAM

This defines the required parameter to configure Cam Core.

```
typedef struct
{
        T_CAMCORE_MODE mode;
        T_CAMCORE_THRESHOLD fifothreshold;
        BOOL vsynch; /* if vsynch = 0, VSYNCH is not available, which is the case for Locosto */
        T_CAMCORE_CCPMODE ccpmode;
        UINT16 xclk_div; /* CAM_XCLK = CAM_MCLK/ xclk_div */

} T_CAMCORE_CONFIGPARAM;
```

2.3.8 T_CAMCORE_REVISION

This defines the required parameter for scaling and cropping operation.

```
typedef struct
{
        INT8 major_rev;
        INT8 minor_rev;
} T_CAMCORE_REVISION;
```

2.4 Camcore_hwapi.h

This file contains parameters related to Camera controller.

THIS FILE SHOULD NOT BE CHANGED FOR DIFFERENT PLATFORMS.

#define **CAMCORE_ISRESETDONE** (CAMCORE_REG (CC_SYSSTATUS));

Return Values

The possible values are:

Value	Definition
0	Internal Module reset is on-going
1	Reset completed

#define **camcore_enableInterrupts** (MASK) {CAMCORE_REG (CC_IRQENABLE) = 0X0000001F;

#define **camcore_enableDMA** (CAMCORE_REG (CC_CTRL_DMA) = (CAMCORE_REG (CC_CTRL_DMA) | 0X00000100));

#define **camcore_disableDMA** (CAMCORE_REG (CC_CTRL_DMA) = (CAMCORE_REG (CC_CTRL_DMA) & 0XFFFFFEFF));

#define **CC_FIFO_DEPTH** (CAMCORE_REG (CC_GENPAR) &0x00000007);
Actual FIFO size would be: 2^CC_FIFO_DEPTH

#define **FIFOREAD** (CAMCORE_REG (CC_FIFODATA));

#define **FIFOWRITE** (DATA) (CAMCORE_REG (CC_FIFODATA) = DATA);

USAGE: data = FIFOREAD;
FIFOWRITE (data);

2.5 Configuration Items

DATA COMING FROM CAMERA STUB:

The following section describes the data rate achieved for camera in viewfinder mode.

Pixel clock input = 6.5 MHz.
Stub is sending data of: 176*2 bytes/line

HSYNCH HIGH = (1/6.5)*176*2 = 54 MICRO SEC.
HSYNCH LOW = (1/6.5)*160 = 24.6 MICRO SEC

AS VSYNCH is always high:
So for a QCIF IMAGE: TIME FOR ONE FRAME = 144 * 78.6 MICRO SEC = 11.318 msec.

Which is equivalent = 88 frames/sec.

LCD

LCD is running at 13 MHz.

DMA CONFIGURATION

DMA is configured from peripheral to peripheral transfer mode.

SUMMARY

Camera configuration with the support of DMA having peripheral to peripheral capability, and with minimum overhead from the camera software, we have achieved the frame rate of 88 frames/sec.

2.6 Limitations

None.

Chapter 3 IMAGE SERVICE

3.1 Introduction

This document describes the API of the GPF IMG Services. The Camera Application (CAMA) and MMI use IMG services for the following purposes:

- JPEG encoding
- JPEG decoding
- Image Processing

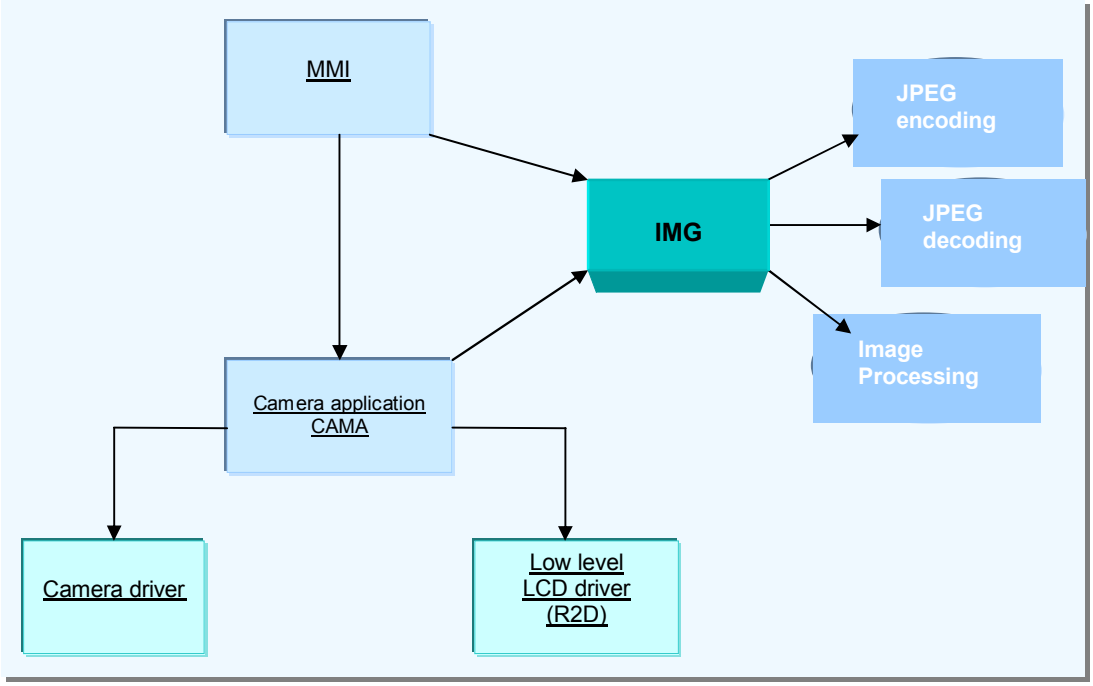


Figure 3 CAMA and IMG Services

IMG Service will use the Emuzed's JPEG encoder on ARM architecture based processors. The Service encodes one raw image or partial image (if streaming mode flag is enabled) in YUV (4:2:2) format to JPEG format.

Similarly, for decoding purpose, IMG Service will use the Emuzed's JPEG decoder. The Service decodes the encoded image in JPEG standard format to YUV (4:2:2) formats.

In order to display the image in LCD driver, IMG should be able to convert the stored image of VGA-YUV format to QCIF-RGB format. To do that first it needs to have a scale down from VGA to QCIF followed by color space conversion from YUV (4:2:2) to RGB (5:6:5).

The API entries honor return path concept, which means that the client can be notified of the result of a service requested by

- Receiving a message in its mail box.
- Using a call back function.

When using a call back function, the response to the service request is passed as a parameter to this function.
If a message is received from IMG, the client is responsible for releasing any associated memory.

3.2 Interface description Application

In the following sections the APIs of IMG Service are described. The IMG service is applicable for Calypso+ (CHIPSET=12) and Locosto (CHIPSET=15). However, there may be few differences in the behavior of the APIs between Locosto and Calypso, which have been explained below.

3.2.1 img_abort (Applicable for both Cal+ and Locosto)

```
T_IMG_RETURN img_abort (T_IMG_RETURN operation_id);
```

Description

It marks an img operation as aborted.

Parameters

- **operation_id**

It is the id of the operation that is to be aborted.

Immediate Return

- **T_IMG_RET**

The possible values are:

id	Definition
IMG_OK	The API function was successfully executed. Expect status message.
IMG_NOT_SUPPORTED	The function is not supported.
IMG_MEMORY_ERR	Insufficient memory to create the context.
IMG_ABORTED	The function is aborted due to some internal reason.
IMG_INVALID_ID	The current operation id is not valid.

3.2.2 g_encode_to_ram (Applicable for both Cal+ and Locosto)

```
T_IMG_RETURN img_encode_to_ram (UINT8 *buf,
                                UINT32 buf_size,
                                T_RVF_MB_ID mb_id,
                                T_IMG_IMAGE_FORMAT image_format,
                                T_IMG_ENCODE_PARAM encode_param,
                                T_RV_RETURN_PATH return_path,
                                void *user_data)
```

Description

This function encodes an image of format YUV (4:2:2) to standard JPEG format.

Parameters

- ***buf**

This parameter is the pointer to the input buffer that holds the YUV image buffer which needs to be encoded.

- **buf _size**

Size of the input buffer.

- **mb_id**

This parameter passes the Memory bank id of the corresponding ENTITY. IMG will allocate buffer either from external memory pool or from internal memory pool.

- **image_format**

Specify the input image format, in present case the input format will be typically YUV (4:2:2).
(Please note that while decoding the image format obtained would be the same as given here for input buffer at the time of encoding).

- **encode_param**

This parameter passes the necessary parameters as a structure T_IMG_ENCODE_PARAM for encoding an image.

- **return_path**

Specifies the return path.

- ***user_data**

Pointer to the user data, if any. If there is no such data, the field takes NULL value.

Immediate Return

- **T_IMG_RET**

The possible values are:

id	Definition
IMG_OK	The API function was successfully executed. Expect status message.
IMG_NOT_SUPPORTED	The function is not supported.
IMG_MEMORY_ERR	Insufficient memory to create the context.
IMG_ABORTED	The function is aborted due to some internal reason.
IMG_INVALID_ID	The current operation id is not valid.

Event Return

If the encoding is successful, a message **IMG_ENCODED_IMAGE** is sent to the calling function depending on the return path specified.
If the encoding is aborted, a message **IMG_ABORT_CFM** is sent through return path.
In case of error, the corresponding error message **IMG_ERROR** is sent to the calling function.

Current restriction of use

None.

3.2.3 img_decode (Applicable only for Locosto)

```
T_IMG_RETURN img_decode (UINT8       *buf,
                         UINT32      buf_size,
                         T_RVF_MB_ID  mb_id,
                         T_IMG_IMAGE_FORMAT image_format,
                         T_IMG_DECODE_PARAM decode_param,
T_RV_RETURN_PATH return_path,
                         void *user_data
                  )
```

Description

This function decodes an image of standard JPEG format to YUV (4:2:2).

Parameters

- ***buf**

This parameter is the pointer to the input buffer that holds the JPEG image that needs to be decoded.

- **buf _size**

Size of the input JPEG buffer.

- **mb_id**

This parameter tells from where memory has to be allocated. (To hold the resultant decoded image, based on SCALING factor IMG would allocate memory to its requirement).

- **image_format**

Specify the input image format, in present case the input format will be JPEG (IMG_FORMAT_JPG). No other format is supported.

- **decode_param**

This parameter passes the necessary parameters as a structure T_IMG_DECODE_PARAM for decoding an image.

- **return_path**

Specifies the return path.

- ***user_data**

Pointer to the user data, if any. If there is no such data, the field takes NULL value.

Immediate Return

- **T_IMG_RET**

The possible values are:

id	Definition
IMG_OK	The API function was successfully executed. Expect status message.

IMG_NOT_SUPPORTED	The function is not supported.
IMG_MEMORY_ERR	Insufficient memory to create the context.
IMG_ABORTED	The function is aborted due to some internal reason.
IMG_INVALID_ID	The current operation id is not valid.

Event Return

If the decoding is successful, a message **IMG_RAW_IMAGE** is sent to the calling function depending on the return path specified.
If the decoding is aborted, corresponding message **IMG_ABORT_CFM** is sent through return path.
In case of error, the corresponding error message **IMG_ERROR** is sent to the calling function.

3.2.4 img_color_convert (Applicable for Locosto)

```
T_IMG_RETURN img_color_convert (T_IMG_YUV_BUFFER      *inputbuf,
                                T_IMG_COLORCONV_PARAM colorconv_param,
                                UINT8 *outbuf,
                                T_RV_RETURN_PATH return_path)
```

Description

This function converts the input YUV (420, 422, 444), Monochrome, RGB444 or YUYV frame to 16-bit (5-6-5) interleaved RGB or 24Bit interleaved RGB format depending on the RGB Format chosen.

Parameters

- ***inputbuf**

This is a pointer pointing the buffer of the input image of YUV (4:2:2) format.

- **colorconv_param**

This parameter passes the necessary parameters as a structure T_IMG_COLORCONV_PARAM necessary for colour conversion of an image.

- ***outbuf**

This parameter points the buffer containing the output image in RGB format.

- **return_path**

Specifies the return path

Immediate Return

- **T_IMG_RET**

The possible values are:

id	Definition
IMG_OK	The API function was successfully executed. Expect status message.

IMG_NOT_SUPPORTED	The function is not supported.
IMG_MEMORY_ERR	Insufficient memory to create the context.
IMG_ABORTED	The function is aborted due to some internal reason.
IMG_INVALID_ID	The current operation id is not valid.

Event Return

Store the converted image to output buffer.
If color conversion is successful, a message **IMG_CHANGED_FORMAT_IMAGE** is sent to the calling function through the return path specified.
If color conversion is aborted, corresponding message **IMG_ABORT_CFM** is sent through return path.
In case of error, the corresponding error message **IMG_ERROR** is sent to the calling function.

Current restriction of use

None.

3.2.5 Img_scale (Applicable for Locosto)

```
T_IMG_RETURN img_scale (T_IMG_YUV_BUFFER    *inputbuf,
                        T_IMG_SCALE_PARAM    scaling_param,
                        T_IMG_YUV_BUFFER     *outbuf,
                        T_RV_RETURN_PATH     return_path
                       )
```

Description

This function performs up-scaling and down-scaling of YUV (420, 422, 444), Monochrome, RGB444 or YUYV frame. The output color format is same as the input color format. Only the cropped region is scaled. In present scenario, the input image format is YUV (4:2:2) only.

Parameters

- ***inputbuf**

This is a pointer pointing the buffer of the input image.

- **scaling_param**

This parameter passes the necessary parameters as a structure T_IMG_SCALE_PARAM necessary for up scaling and downscaling of an image.

- ***outbuf**

This parameter points the buffer containing the output image.

- **return_path**

Specifies the return path

Immediate Return

- **T_IMG_RET**

The possible values are:

id	Definition
IMG_OK	The API function was successfully executed. Expect status message.
IMG_NOT_SUPPORTED	The function is not supported.
IMG_MEMORY_ERR	Insufficient memory to create the context.
IMG_ABORTED	The function is aborted due to some internal reason.
IMG_INVALID_ID	The current operation id is not valid.

Event Return

Store the scaled image to output buffer.
If scaling is successful, a message **IMG_CHANGED_FORMAT_IMAGE** is sent to the calling function through the return path specified.
If scaling is aborted, corresponding message **IMG_ABORT_CFM** is sent through return path.
In case of error, the corresponding error message **IMG_ERROR** is sent to the calling function.

Current restriction of use

None.

3.2.6 Img_rotate (Applicable for Locosto)

```
T_IMG_RETURN img_rotate (T_IMG_YUV_BUFFER    *inputbuf,
                         T_IMG_ROTATION_PARAM rotation_param,
                         T_IMG_YUV_BUFFER     *outbuf,
                         T_RV_RETURN_PATH     return_path
                        )
```

Description

This function performs rotation of YUV (420, 422, 444), Monochrome or RGB444 input buffer by 90 degree, 180 degree or 270 degree as specified by rotation parameter rotate_flag. The output color format is same as input color format except for YUV 422.
If the source is of type YUV422H the output format will be in YUV422V and vice versa for 90degree and 270 degree rotation.

Parameters

- ***inputbuf**

This is a pointer pointing the buffer of the input image.

- **rotation_param**

This parameter passes the necessary parameters as a structure T_IMG_ROTATION_PARAM necessary for rotating an image.

- ***outbuf**

This parameter points the buffer containing the output image.

- **return_path**

Specifies the return path

Immediate Return

- **T_IMG_RET**

The possible values are:

id	Definition
IMG_OK	The API function was successfully executed. Expect status message.
IMG_NOT_SUPPORTED	The function is not supported.
IMG_MEMORY_ERR	Insufficient memory to create the context.
IMG_ABORTED	The function is aborted due to some internal reason.
IMG_INVALID_ID	The current operation id is not valid.

Event Return

Store the rotated image to output buffer.
If scaling is successful, a message **IMG_CHANGED_FORMAT_IMAGE** is sent to the calling function through the return path specified.
If scaling is aborted, corresponding message **IMG_ABORT_CFM** is sent through return path.
In case of error, the corresponding error message **IMG_ERROR** is sent to the calling function.

Current restriction of use

None.

3.2.7 Img_change_format (Applicable for Locosto)

```
T_IMG_RETURN img_change_format (UINT8 *inbuf,
                                UINT32 inbuf_size,
                           T_IMG_CHANGE_FORMAT_PARAM change_format_param,
      T_IMG_YUV_BUFFER     *outbuf,
                           T_RV_RETURN_PATH return_path,
                           void *user_data
                          )
```

Description

This function decodes an image of standard JPEG format to the required output format. In present scenario the output format would be QCIF/RGB (5-6-5).

Typically this function will decode the stored JPEG image to YUV/VGA image, at the first step. Then depending on the output format, it can scale down the VGA image to QCIF format (scaling) followed by colour conversion from YUV (4:2:2) to RGB (5:6:5), if required.

Parameters

- ***inbuf**

This parameter is the pointer to the input buffer that holds the JPEG image that needs to be decoded.

- **inbuf _size**

Size of the input JPEG buffer.

- **change_format_param**

Required parameter necessary to change an image from one format to another, passed as a structure T_IMG_CHANGE_FORMAT_PARAM.

- ***outbuf**

This parameter is the pointer to the output buffer that holds the output image.

- **return_path**

Specifies the return path.

- ***user_data**

Pointer to the user data, if any. If there is no such data, the field takes NULL value.

N.B. The IMG will allocate the buffer to hold intermediate image from memory allocated to CAMD.

Immediate Return

- **T_IMG_RET**

The possible values are:

id	Definition
IMG_OK	The API function was successfully executed. Expect status message.
IMG_NOT_SUPPORTED	The function is not supported.
IMG_MEMORY_ERR	Insufficient memory to create the context.
IMG_ABORTED	The function is aborted due to some internal reason.
IMG_INVALID_ID	The current operation id is not valid.

Event Return

If the process is successful, a message **IMG_CHANGED_FORMAT_IMAGE** is sent to the calling function through the return path specified.
If the process is aborted, corresponding message **IMG_ABORT_CFM** is sent through return path.
In case of error, the corresponding error message **IMG_ERROR** is sent to the calling function.

Current restriction of use

Output image format supports only RGB (5-6-5).

3.3 Message definition

There are two types of messages, request messages and response messages. All the request message definitions contain return path and the response message structures contain operation id as their status information. The message definitions are located in the directory Img_message.h.

3.3.1 IMG_ENCODE

This request message is sent to IMG ENTITY containing all the necessary information regarding encoding of the input image.

```
typedef struct
{
        T_RV_HDR                hdr;
        UINT8                   *buf;
        T_RVF_MB_ID             mb_id;
        T_IMG_IMAGE_FORMAT      image_format;
        T_IMG_ENCODE_PARAM      encode_param;
        T_IMG_RETURN            operation_id;
        T_RV_RETURN_PATH        return_path;
        void                    *user_data;
} T_IMG_ENCODE;
```

3.3.2 IMG_DECODE

This request message is sent to IMG ENTITY containing all the necessary information regarding decoding of the input image.

```
typedef struct
{
        T_RV_HDR                hdr;
        UINT8                   *buf;
        UINT32                  buf_size;
        T_RVF_MB_ID             mb_id;
        T_IMG_DECODE_PARAM      decode_param;
        T_IMG_IMAGE_FORMAT      image_format;
        T_RV_RETURN_PATH        return_path;
        T_IMG_RETURN            operation_id;
        void                    *user_data;
} T_IMG_DECODE;
```

3.3.3 IMG_COLORCONV

This request message is sent to IMG ENTITY containing all the necessary information regarding conversion of the colour format of the input image.

```
typedef struct
{
        T_RV_HDR                        hdr;
        T_IMG_YUV_BUFFER                *inbuf;
        T_IMG_COLORCONV_PARAM           colorconv_param;
        T_IMG_YUV_BUFFER        *outbuf;
        T_IMG_RETURN                    operation_id;
        T_RV_RETURN_PATH                return_path;
        void                            *user_data;

} T_IMG_COLORCONV;
```

3.3.4 IMG_SCALING

This request message is sent to IMG ENTITY containing all the necessary information regarding up scaling or down scaling of the image format of the input image.

```
typedef struct
{
        T_RV_HDR                hdr;
        T_IMG_YUV_BUFFER        *inbuf;
        T_IMG_SCALE_PARAM       scaling_param;
        T_IMG_YUV_BUFFER        *outbuf;
        T_IMG_RETURN            operation_id;
        T_RV_RETURN_PATH        return_path;
        void                    *user_data;

} T_IMG_SCALING;
```

3.3.5 IMG_ROTATION

This request message is sent to IMG ENTITY containing all the necessary information regarding rotation of the image format of the input image.

```
typedef struct
{
        T_RV_HDR                hdr;
        T_IMG_YUV_BUFFER        *inbuf;
        T_IMG_ROTATION_PARAM    rotation_param;
        T_IMG_YUV_BUFFER        *outbuf;
        T_IMG_RETURN            operation_id;
        T_RV_RETURN_PATH        return_path;
        void                    *user_data;

} T_IMG_ROTATION;
```

3.3.6 IMG_CHANGE_IMAGE

This request message is sent to IMG ENTITY containing all the necessary information regarding change the image format of the input image.

```
typedef struct
{
        T_RV_HDR                hdr;
        UINT8                   *inbuf;
        UINT32                  inbuf_size;
        T_IMG_CHANGE_FORMAT_PARAM change_format_param;
        T_IMG_YUV_BUFFER        *outbuf;
        T_IMG_RETURN            operation_id;
        T_RV_RETURN_PATH        return_path;
        void                    *user_data;

} T_IMG_CHANGE_IMAGE;
```

3.3.7 IMG_RAW_IMAGE

This message sends a positive response to the IMG stating decoding is successful, and points to the decoded image by *image_list_p.

```
typedef struct
{
        T_RV_HDR            hdr;
        T_IMG_IMAGE_LIST    *image_list_p;
        T_IMG_RETURN        operation_id;
        void                *user_data;

} T_IMG_RAW_IMAGE;
```

3.3.8 IMG_ ABORT_CFM

This message sends a response to the IMG when the current image is aborted.

```
typedef struct
{
        T_RV_HDR                hdr;
        T_IMG_RETURN            operation_id;

} T_IMG_ABORT_CFM;
```

3.3.9 IMG_ERROR

Message to tell service user that current operation is unsuccessful.

```
typedef struct
{
        T_RV_HDR                hdr;
        T_IMG_RETURN            operation_id;
        T_IMG_RESULT_CODE       error_code;
        void                    *user_data;

} T_IMG_ERROR;
```

3.3.10 IMG_ENCODED_IMAGE

This message sends a positive response to the IMG stating encoding is successful, and points to the encoded image by *encoded_data_p.

```
typedef struct
{
        T_RV_HDR                hdr;
        T_RVF_BUFFER            *encoded_data_p;
        UINT32                  encoded_size;
        T_IMG_RETURN            operation_id;
        void                    *user_data;
```

```
} T_IMG_ENCODED_IMAGE;
```

3.3.11 IMG_CHANGED_FORMAT_IMAGE

This message sends a positive response to the IMG ENTITY stating the necessary modification is successful, and points to the modified image by *changed_data_p.

```
typedef struct
{
        T_RV_HDR                hdr;
        T_IMG_RETURN            operation_id;
        T_RVF_BUFFER            *output_data_p;
        Void                    *user_data;

} T_IMG_CHANGED_IMAGE;
```

3.4 Types definitions and constants

API type definitions and constants are located in the configuration file img_api.h in the common directory.

3.4.1 T_IMG_RETURN

Currently they are the standard RV return types, but they may be customized in the future.

```
typedef T_RV_RET T_IMG_RETURN;

#define IMG_OK                  RV_OK
#define IMG_NOT_SUPPORTED       RV_NOT_SUPPORTED
#define IMG_MEMORY_ERR          RV_MEMORY_ERR
#define IMG_ABORTED             -100
#define IMG_INVALID_ID          -101
```

3.4.2 T_IMG_IMAGE_LIST

Specifies the properties of an image.

```
typedef struct T_IMG_IMAGE_LIST_TAG {
        UINT8   *lum;
        UINT8    *cb;
        UINT8    *cr;
        UINT16   image_width;
        UINT16   image_height;
        UINT32   duration;
        BOOLEAN loop;
        void *next_image;        //Pointer to next T_IMG_IMAGE_LIST_TAG
} T_IMG_IMAGE_LIST;

typedef T_IMG_IMAGE_LIST *T_IMG_IMAGE_LIST_PTR;
```

3.4.3 T_IMG_IMAGE_FORMAT

The following structure specifies the possible formats that an image may have

```
typedef enum IMG_IMAGE_FORMAT_TAG{
        IMG_FORMAT_JPG = 1,
        IMG_FORMAT_BMP,
        IMG_FORMAT_GIF,
        IMG_FORMAT_wBMP,
        IMG_FORMAT_PNG,
        IMG_FORMAT_RAW_YUYV,
        IMG_FORMAT_EMPTY
} T_IMG_IMAGE_FORMAT;
```

3.4.4 T_IMG_RESULT_CODE_TAG

Defines all possible results for an IMG operation.

```
typedef enum IMG_IMAGE_FORMAT_TAG{
        IMG_SUCCESS                     = 1,
        IMG_ERROR_ABORTED               = -1,
        IMG_ERROR_FORMAT_UNKNOWN        = -2,
        IMG_ERROR_IMAGE_CORRUPT         = -3,
        IMG_ERROR_INT_MEMORY            = -4,
        IMG_ERROR_EXT_MEMORY            = -5
} T_IMG_RESULT_CODE;
```

3.4.5 T_IMG_ENCODING_MODE

Defines the possible encoding modes.

```
typedef enum {
        IMG_JPEG_BASELINE=0,
        IMG_JPEG_PROGRESSIVE =1
} T_IMG_ENCODING_MODE;
```

IMG_JPEG_PROGRESSIVE is not supported presently.

3.4.6 T_IMG_ENCODE_PARAM

Defines the parameter for encoding.

(For Calypso+)

```
typedef struct {
        UINT16 max_x;
        UINT16 max_y;
        T_IMG_ENCODING_MODE encoding_mode;
        UINT16 precision; /* bits per channel */
        UINT16 quality_factor; /* Q-factor (JPEG encoding) */
} T_IMG_ENCODE_PARAM;
```

(For Locosto)

```
        typedef struct {
        uint32 max_x;
        uint32 max_y;
        T_IMG_ENCODING_MODE encoding_mode;
        uint32 quality_factor;          /* Q-factor (JPEG encoding) */
        uint32 streaming_mode;
} T_IMG_ENCODE_PARAM;
```

- **max_x:** Maximum dimension value in horizontal direction among all the components of the input raw image.

- **max_y:** Maximum dimension value in vertical direction among all the components of the input raw image.

- **Precision**: It indicates bits per channel.

- **encoding mode**: In our case, only BASELINE MODE is supported.

- **quality_factor:** This parameter is used to control the quality of encoding. It can take values from 1 to 100. Quality factor 1 produces least quality and 100 gives best quality. If the value of input quality_factor is out of the range (1, 100) its value shall be forced to 50 internally.

- **streaming_mode:** This flag is to indicate whether encoding needs to be done in one shot (if the flag is set to 'E_OFF') for whole image or in multiple times (if the flag is set 'E_ON') based on the available output buffer size. However the output buffer size must be at least enough to hold bytes to encode one MCU row. As the API does not support the encoding of partial image, streaming_mode should always be 0.

3.4.7 T_IMG_DECODE_PARAM

Defines the parameter for decoding

```
        typedef struct {
        UINT32 x_offset;
        UINT32 y_offset;
        UINT8   sampling_factor;
        UINT8   num_rows;
        UINT8   scaling_factor;
} T_IMG_DECODE_PARAM;
```

- **x_offset**: [IN] Crop window start position in the source frame.

- **y_offset**: [IN] Crop window start position in the source frame.

- **sampling_factor**: [IN] indicates the sampling factor that determines the relation between Y and Cr, Cb components. For YUV (422) image (which is our present case) it should be 1.

- **num_rows**: [IN] Indicates the number of MCU rows to be decoded for baseline JPEG image. For the entire image to be decoded, the typical value of num_rows is 0.

- **scaling_factor**: [IN] indicates the scale down factor in DCT domain. The supported values are 1, 2, 4 and 8 only. Any other value passed will return E_ERROR_ARGUMENT.

3.4.8 T_IMG_COLORCONV_PARAM

This defines the required parameter for color conversion operation.

```
typedef struct{
        UNIT16 actWidth;
        UINT16 actHeight;
        UINT8  srcClrFfmt;
        int32    numBytes;
        UINT8  destClrFmt;
} T_IMG_COLORCONV_PARAM;
```

- **actWidth :** [IN] Actual width of the image

- **actHeight**: [IN] Actual height of the image.

- **srcClrFmt**: [IN] Color format of the source buffer. The possible color formats are

Color Formats	Corresponding values
YUV420	0x01
YUV422H	0x02
YUV422V	0x03
YUV444	0x04
RGB444	0x05
MONOCHROME	0x06
YUYV	0x07

- **numBytes** : [IN] The number of Bytes in a row of RGB buffer. This is required for RGB24 output format as some display devices expect each row of data to be word aligned.

- **destClrFmt**: [IN] Color format of the destination (0 - RGB565 and 1 - RGB24).

3.4.9 T_IMG_SCALE_PARAM

This defines the required parameter for scaling and cropping operation.

```
typedef struct{
```

```
UINT32 x_offset;
UINT32 y_offset;
UINT32 cropWidth;
UINT32 cropHeight;
UINT32 srcClrFmt;
}T_IMG_SCALE_PARAM;
```

- **x_offset** : [IN] Crop window start position in the source frame.
- **y_offset** : [IN] Crop window start position in the source frame.
- **cropWidth** : [IN] Width of the crop window.
- **cropHeight** : [IN] Height of crop window.
- **srcClrFmt**: [IN] specifies the color format of the input video. Possible color formats are.

Color Formats	Corresponding values
YUV420	0x01
YUV422H	0x02
YUV422V	0x03
YUV444	0x04
RGB444	0x05
MONOCHROME	0x06
YUYV	0x07

3.4.10 T_IMG_ROTATION_PARAM

This defines the required parameter necessary for rotation operation.

```
typedef struct{
        UINT8           rotate_flag;
        UINT32          srcClrFmt;
} T_IMG_ROTATION_PARAM;
```

- **rotate_flag**: this specifies the angle of rotation.

0: No rotation
1: 90deg rotation
2:180deg rotation
3:270deg rotation

- **srcClrFmt:** Specifies the colour format of the input image. It also returns the output colour format after rotation.

3.4.11 T_IMG_CHANGE_FORMAT_PARAM

This defines the required parameter for changing the encoded image to a standard image format that is valid for LCD display.

```
typedef struct {
        T_IMG_IMAGE_FORMAT input_image_format;
        T_IMG_DECODE_PARAM decode_param;
        T_IMG_COLOR_FORMAT output_format;
        } T_IMG_CHANGE_FORMAT_PARAM;
```

- **input_image_format**: the format of the input image that is subjected to be changed. In present case the input image format is JPEG.

- **decode_param:** necessary parameters for decoding the input image.

- **Output_format:** the format of the output image. This will contain resolution, destination colour format etc. of the final image.

3.4.12 T_IMG_COLOR_FORMAT

The structure defines a colour format as well as its resolution of an image.

```
typedef struct{
        T_IMG_RESOLUTION   resolution;
        UINT16             x_length;
        UINT16             y_length;
        T_IMG_FORMAT       color_format;
        int32              numBytes;
        } T_IMG_COLOR_FORMAT;
```

- **resolution**: defines the resolution of the image. Possible values of the resolution are defined by the enum T_IMG_RESOLUTION, i.e., VGA, QVGA, CIF, and QCIF.

- **x_length**: number of pixels per line in a given resolution.

- **y_length**: number of lines that a given resolution can support.

- **color_format**: defines the colour format of the image. The possible colour formats are defined by the enum T_IMG_FORMAT.

- **numBytes**: [IN] The number of Bytes in a row of RGB buffer. This is required for RGB24 output format as some display devices expect each row of data to be word aligned.

3.4.13 T_IMG_RESOLUTION

Defines all possible resolutions:

```
typedef enum IMG_RESOLUTION{
        IMG_VGA=1,
        IMG_QVGA,
        IMG_CIF,
        IMG_QCIF
        } T_IMG_RESOLUTION;
```

3.4.14 T_IMG_FORMAT

Defines all possible colour formats:

```
typedef enum IMG_FORMAT{
        IMG_RGB565=0,
        IMG_RGB444,
        IMG_RGB666,
        IMG_RGB888,
        IMG_YUV444,
        IMG_YUV422,
        IMG_YUV420,
        } T_IMG_FORMAT;
```

3.4.15 T_IMG_YUV_BUFFER

Defines the structure of a YUV buffer. If the buffer does not contain YUV image format, then only l*lum component will point to the required data.

```
typedef struct  IMG_YUV_BUFFER{
        UINT8  *lum;
        UINT8  *cb;
        UINT8  *cr;
        UITN16 width;
        UINT16 height;
        UINT8   color_format;
} T_IMG_YUV_BUFFER;
```

- ***lum:** pointer to the buffer for storing the luminance component.
- ***cb:** pointer to the buffer for storing the Cb (chrominance) component.
- ***cr:** pointer to the buffer for storing the Cr (chrominance) component.
- **Height:** Height of the luminance frame buffer.
- **Width:** Width of the luminance frame buffer.

3.5 Configuration Items

None

A.

Chapter 4 LCD DRIVER

/* LCD DRIVER COMPLETELY CHANGED */
/* CONTACT VAIDY */

4.1 Purpose of the Document

This document describes the new APIs that are proposed for Locosto LCD driver. These APIs provide basic LCD functionalities and don't provide any graphics features.

4.2 Overview

The current LCD driver supports two LCD's .Earlier , driver used R2D heavily . But due to high memory requirements by R2D , now R2D is used only by RTEST (For test build) .

4.3 System Overview

The LCD driver is memory optimized .This LCD driver directly interacts with MMI . The LCD Manager provides an abstraction for the different LCDs that might be present in the hardware. The LCD interface layer abstracts the different interfaces through which an LCD could be connected in the hardware.

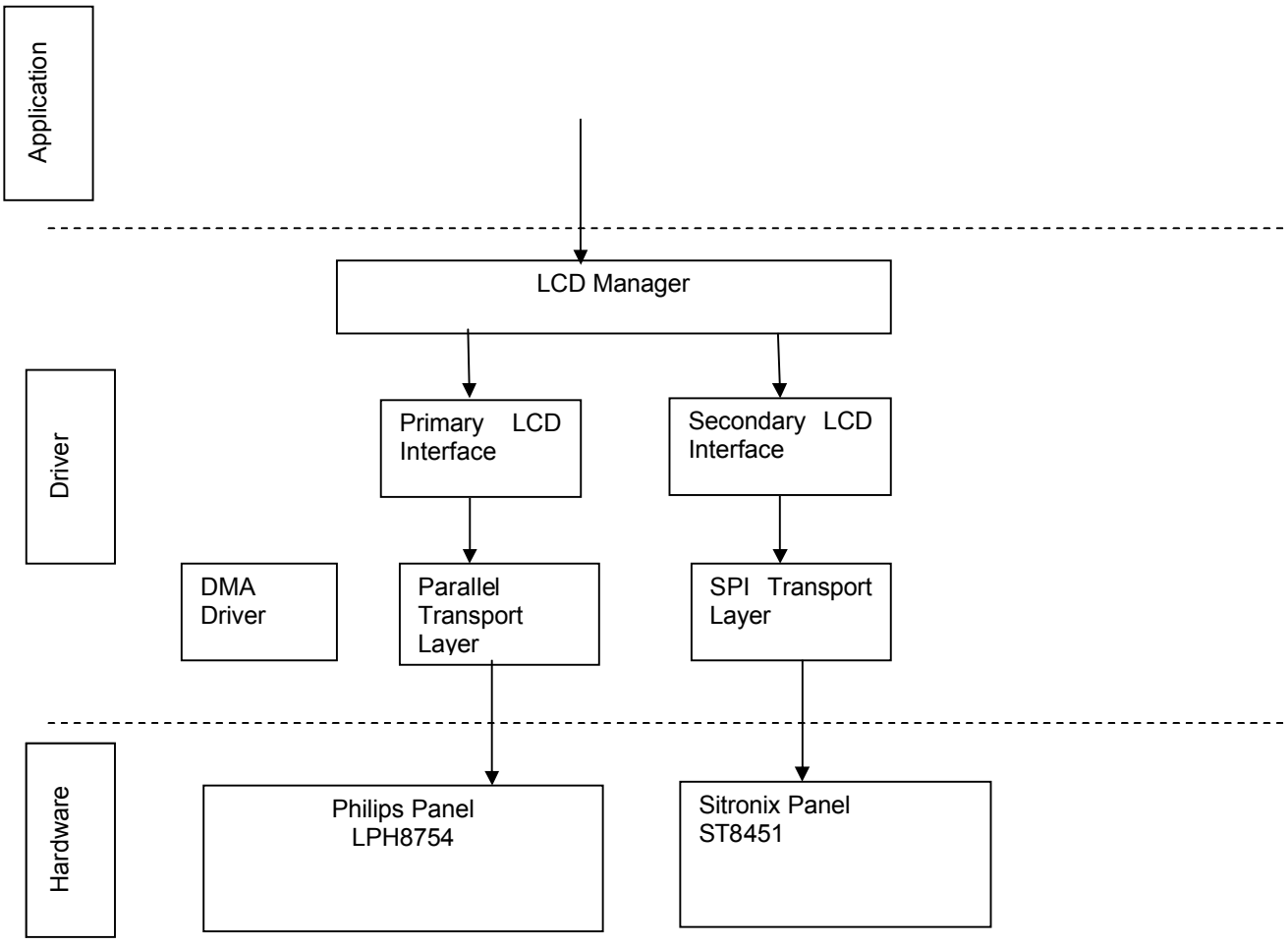


Figure 4 Design of LCD Driver

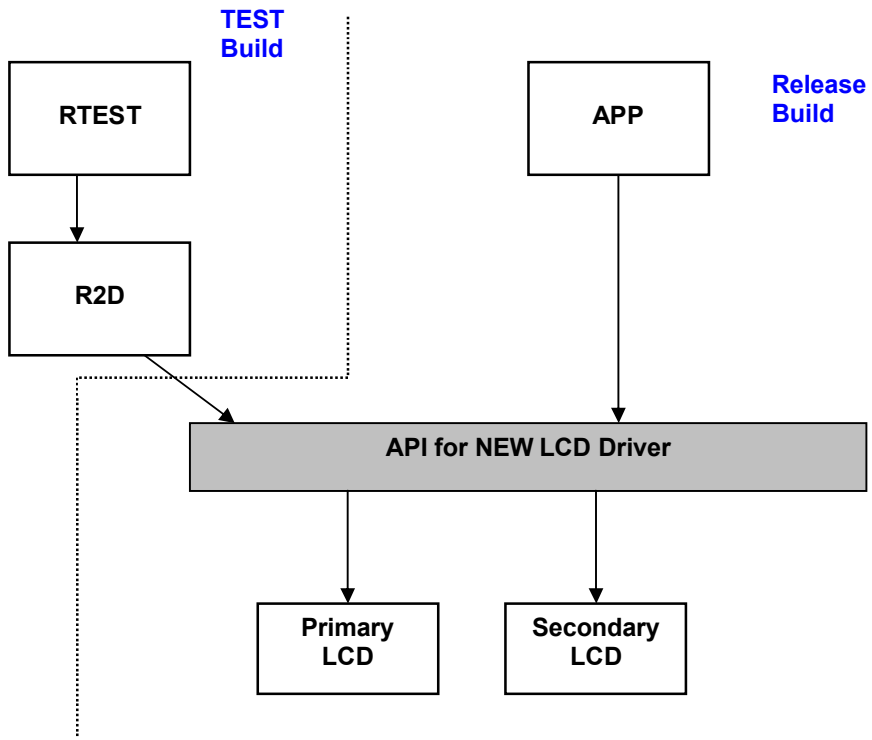


Figure 5 Architecture for LCD Driver

The API for the LCD driver will facilitate the following:
1) Uniform Interface for the Application / MMI
2) Handle different interfaces and panels possible
3) Facilitate configuration of LCD parameters like endianness, RGB format, etc.,

4.4 Interface Description

The following section describes the interface APIs from the LCD driver.

There are only 3 APIs:
1. lcd_init – Initialisation of the LCD interface, controller and display
2. lcd_display – for dumping the framebuffer data received onto the LCD display
3. lcd_control – control of the LCD display.

4.5 API Description

4.5.1 lcd_initialization

```
T_RV_RET lcd_initialization(T_LCD_SELECT sel)
```

Description

This function initialises the LCD display and the LCD controller driver.This API should be called before any other functions in this driver.

Parameters

- **sel** specifies whether this API is intended for main or sub-LCD.

Immediate Return

- **T_RV_RET**

The possible values are:

id	value	Definition
RV_OK	0	
RV_NOT_SUPPORTED	**-2**	
RV_NOT_READY	**-3**	
RV_MEMORY_WARNING	**-4**	
RV_MEMORY_ERR	**-5**	
RV_MEMORY_REMAINING	**-6**	
RV_INTERNAL_ERR	**-9**	
RV_INVALID_PARAMETER	**-10**	

Event Return

Current restriction of use

The API should be called before any other functions in this driver.

4.5.2 lcd_display

```
T_RV_RET lcd_display(lcdSelect sel, Uint16 *imageDataptr, lcd_fb_coordinates *p_lcd_coord)
```

Description

This function loads the pixel data only, from the frame buffer onto the LCD display. This function alone will be part of the "**lcd refresh**" task. Once the refresh operation is complete, the client will be sent a message. This API could take care of multiple requests and respond accordingly.

Parameters

- **sel**

specifies whether this API is intended for main or sub-LCD.

- **imageDataPtr**

Pointer to the image data to be transferred. The size of the pixel data buffer is assumed to be for the entire LCD screen.

- **p_lcd_coord**

Structure where the LCD pixel co-ordinates are specified for the start and end position.

Immediate Return

- **T_RV_RET**

The possible values are:

id	value	Definition
RV_OK	0	
RV_NOT_SUPPORTED	-2	
RV_NOT_READY	-3	
RV_MEMORY_WARNING	-4	
RV_MEMORY_ERR	-5	
RV_MEMORY_REMAINING	-6	
RV_INTERNAL_ERR	-9	
RV_INVALID_PARAMETER	-10	

Event Return

None

Current

restriction of use

lcd_init, lcd_config should have been called once before this call.

4.5.3 lcd_control

```
T_RV_RET lcd_control(lcdSelect sel, T_LCD_COMMAND command, void
*p_cmd_param)
```

Description

This is a generic API which could be further scaled for any additional commands which might come up later. Currently added commands are listed below:

Command Description

- **LCD_GETCONFIG** Need to pass a structure pointer of type “lcd_tuningtable” to get the configuration items.

- **LCD_SETCONFIG** Need to pass a structure pointer of type “lcd_configparams” to set the configuration items.
- **LCD_DISPLAYON** Only command is sufficient. Display is switched ON.
- **LCD_DISPLAYOFF** Only command is sufficient. Display is switched OFF.
- **LCD_CLEAR** Only command is sufficient. Contents of the LCD are cleared.

Parameters

- **sel** specifies whether this API is intended for main or sub-LCD.

- **Command** Command that can be given to the LCD driver for eg., clear, Display On, Display OFF, etc.

- **p_cmd_param** Structure to pass parameters if required for the commands.

Immediate Return

- **T_RV_RET**

The possible values are:

id	value	Definition
RV_OK	0	
RV_NOT_SUPPORTED	**-2**	
RV_NOT_READY	**-3**	
RV_MEMORY_WARNING	**-4**	
RV_MEMORY_ERR	**-5**	
RV_MEMORY_REMAINING	**-6**	
RV_INTERNAL_ERR	**-9**	
RV_INVALID_PARAMETER	**-10**	

Event Return

None.

Current restriction of use

None.

4.6 Types definitions and constants

4.6.1 T_LCD_SELECT

T_LCD_SELECT selects a specific LCD

Synopsis
```
typedef enum {
        DISPLAY_MAIN_LCD,
        DISPLAY_SUB_LCD
} T_LCD_SELECT;
```

4.6.2 T_LCD_ENDIAN

T_LCD_ENDIAN selects the endianness to be used for the Pixel data

Synopsis
```
typedef enum {
        LITTLE_ENDIAN,
        BIG_ENDIAN
} T_LCD_ENDIAN;
```

4.6.3 T_LCD_PIXFORMAT

T_LCD_PIXFORMAT selects the Pixel format for the pixel data

Synopsis
```
typedef enum {
        RGB565,
        RGB666,
        RGB888
                } T_LCD_PIXFORMAT;
```

4.6.4 T_LCD_ORIENTATION

T_LCD_ORIENTATION selects the orientation of the LCD

Synopsis
```
typedef enum {
        HORIZONTAL,
        VERTICAL
                } T_LCD_ORIENTATION;
```

4.6.5 T_LCD_REFCONTROL

T_LCD_REFCONTROL selects if LCD refresh is enabled or disabled

Synopsis
```
typedef enum {
        REF_ENABLED,
        REF_DISABLED
                } T_LCD_REFCONTROL;
```

4.6.6 T_LCD_COMMAND

T_LCD_COMMAND	Command to be issued to the LCD driver

Synopsis

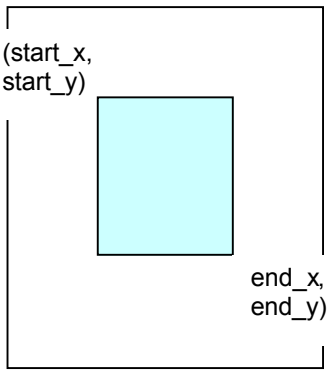
```
typedef enum {
        LCD_CLEAR,
        LCD_DISPLAYON,
        LCD_DISPLAYOFF,
        LCD_GETCONFIG,
        LCD_SETCONFIG
            } T_LCD_COMMAND;
```

4.6.7 lcd_fb_coordinates

Lcd_fb_coordinates	selects the Pixel co-ordinates to be refreshed

Synopsis

```
typedef struct {
        Uint16 start_x;
        Uint16 start_y;
        Uint16 end_x;
        Uint16 end_y;
            } lcd_fb_coordinates;
```



Window area to be refreshed

4.6.8 lcd_configparams

lcd_configParams	Parameters for LCD configuration. These are the parameters which could be configured from the application.

Synopsis typedef struct {
Uint16 **height** /* height of the display panel */
Uint16 **width** /* width of the display panel */
T_LCD_ORIENTATION **orientation** /* orientation of the LCD */
T_LCD_PIXFORMAT **pixel_format**; /* RGB format */
T_LCD_ENDIAN **endianness** /* Endianness of the pixel data */
T_LCD_REFCONTROL **refresh_control** /* refresh control */
} **lcd_configparams;**

4.6.9 lcd_tuningtable

lcd_TuningTable parameters of the tuning table. This table gives the whole list of parameters which the application can configure as well as the read-only parameters which are controlled at the driver level.

Synopsis Typedef struct {
bool **partial_update**; /* does it support windowing or partial update of the LCD framebuffer*/
bool **OSD**; /* does it support OSD (On Screen Display */
bool **dedicated_dma**; /* is there dedicated dma */
lcd_configparams ***p_lcd_configparams**
}**lcd_tuningtable;**

Chapter 5 DMA CONTROLLER

5.1 Introduction

The Direct Memory Access driver (DMA) is a new ENTITY. It is created to support the new DMA controller hardware device of the Locosto chipset. This document describes the application interface of the DMA SWE.

The main task of the DMA SWE is the management of the available DMA channels.
The DMA ENTITY can be seen as a high level driver (the DMA driver with the API), a low-level DMA driver, and the GPF components HISR + generic functions.

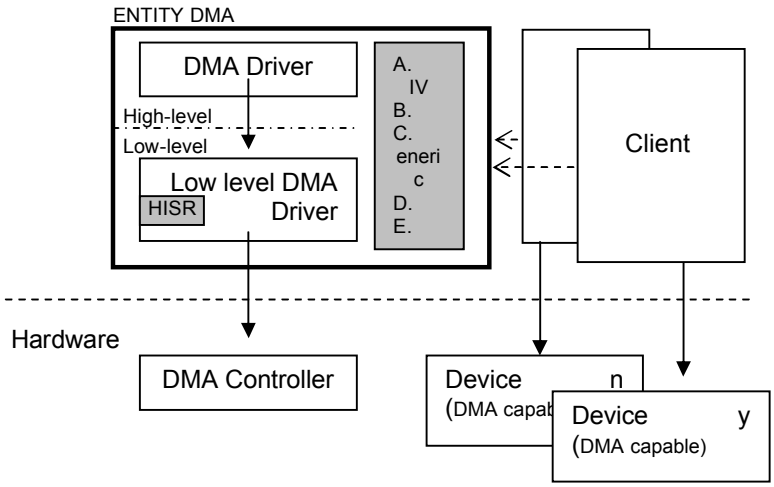


Figure 6 API of DMA

The DMA ENTITY is developed as a GPF compatible entity . The ENTITY is not using (depending on) other ENTITIES. All API's (with the exception of dma_get_sw_version()) are accessible through function call and GPF messages.

5.2 Return Mechanism

All the functions return an immediate value, providing information on the success or the failure of the function call. In some cases, extra processing time might be needed to perform the action requested when calling the function. In this case, the function is exit and later on, one or several MESSAGEs are sent back by the DMA entity.

The DMA entity use the MESSAGE format and the return path method defined in GPF Environment. Basically, in order to send information back, the DMA entity sends MESSAGEs to the client. A MESSAGE is a buffer, with a header, common to any MESSAGE, and a custom field related to the MESSAGE. The header is a C structure, containing the *msg_id* field. This field contains the unique msg_id of the MESSAGE and is the only way to know which kind of MESSAGE has been received. Based on this value, the client can re-cast the buffer and access to custom information related to the MESSAGE.

Clients have two ways to get access to the MESSAGEs:
Call back functions or message posted with its ADDRESS ID.

A call back function is a function name, provided by client as a parameter and will be called by the DMA SW when a MESSAGE occurs. When a callback function is defined, it is always the callback function mechanism that is used to return MESSAGE to the client.

But, for more efficient implementation it also possible to directly send a message to the client. In this case, the ADDR ID of the client must be provided to the DMA entity. That implies that the client is a GPF entity.

The client can define which return mechanism should be used. For that purpose, it must provide a *return_path*. The generic *return_path* type is a C structure, defined as:

```
typedef struct {
T_RVF_ADDR_ID                       addr_id;
VOID                               (*callback_func)(void *);
} T_RV_RETURN_PATH;
```

This chapter is used for the ENTITY interface description. It is not required to specify the Generic interface.

5.3 Service functions definition

5.3.1 dma_reserve_channel

```
T_RV_RET dma_reserve_channel (      T_DMA_SPECIFIC    specific,
T_DMA_CHANNEL                       channel,
                                    T_DMA_QUEUE       queue,
                                    T_DMA_SW_PRIORITY sw_priority,
T_RV_RETURN                         return_path)
```

Description

This function allows the reservation of a free DMA channel or of a specific channel.
If the request is honoured, a channel number shall be returned as a result. This channel number is required for other function calls like programming DMA transfer information and to enable the transfer.

A limited number of channels are available. If the request can not be granted at the time, it can be queued as an option. When a channel comes available later, waiting reservation requests are handled with respect to the given software priority.

The function returns immediately and handles the request asynchronously. The message is then validated and handled. The return path is used to inform the client about the result of the message processing and to inform the client of any asynchronous events.
An example of an asynchronous event is a status message informing the client that the reservation request is granted at a later time.

Parameters

- **specific**

specific indicates whether the client request a specific channel or the first available.

- **channel**

The *channel* number if the client request an specific channel (see parameter *specific*). If the request is not for a specific channel, this parameter is ignored.
Channel range is from DMA_MIN_CHANNEL to DMA_MAX_CHANNEL.

- **queue**

queue indicates if the request is to be queued when it can not be granted immediately.

- **sw_priority**

The *sw_priority* number is used when a reservation is handled from the queue. The reservation with the highest sw_priority (lowest number) is handled first. When several reservations are made with the same sw_priority, the reservations are handled on a first-in/first-out basis.

If the request is not queued, this parameter is ignored.

sw_priority range is from DMA_SW_PRIORITY_LOWEST to DMA_SW_PRIORITY_HIGHEST.

- **return_path**

Return path for notifications See 5.2 for a description.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed. Expect status message.
RV_MEMORY_ERR	Insufficient memory to create the context.
RV_NOT_READY	The driver is not able to handle this request at this moment. (SWE initialization is not done correctly).

Event Return

- **DMA_STATUS_RSP_MSG**

This message is send to the client to return the status and result of the requested action. Other API's also uses this message to return status or to notify clients of asynchronous events.

structure member ***result.status***

For this action, the value of the message structure member ***result.status*** can have the following values:

Message	Definition
DMA_RESERVE_OK	Request is granted. The structure member result. Channel holds the granted channel number.
DMA_QUEUED	Request could not be granted now and is queued. The structure member result. channel holds a channel queue identifier that can be used to remove the reservation from the queue (with the function dma_remove_from_queue()).
DMA_NO_CHANNEL	Request denied because there is no free channel available.
DMA_TOO_MANY_REQUESTS	Request could not be granted and the queue is full.
DMA_INVALID_PARAMETER	One ore more of the parameters is incorrect.
DMA_NOT_READY	Requested process is supported but cannot be processed now. (SWE initialization is not done correct).
DMA_MEMORY_ERR	The available memory within the DMA SWE is insufficient to process the command.

structure member ***result.channel***

For this action, the value of the message structure member ***result.channel*** holds only a valid value if the member ***result.status*** *has the value* DMA_RESERVE_OK or DMA_QUEUED. For DMA_RESERVE_OK the member holds the assigned channel number. For DMA_QUEUED the member holds a channel queue identifier that can be used by the client if he wants to remove the request from the queue (using the function *dma_remove_from_queue()).*

Note that this API functionality can also be invoked with the message DMA_RESERVE_CHANNEL_REQ_MSG.

Current restriction of use

None

5.3.2 dma_remove_from_queue

```
T_RV_RET dma_remove_from_queue(T_DMA_CHANNEL    channel_queue_id)
```

Description

This function allows the removal of the queued channel reservation request.

The function returns immediately and handles the request asynchronously. The message is then validated and handled. The return path is used to inform the client about the result of the message processing and to inform the client of any asynchronous events.

If the client is not queued, the message status DMA_ACTION_NOT_ALLOWED is returned.
If it was queued, the message status DMA_OK is returned.

Parameters

- **channel_queue_id**

This parameter is the returned queue identifier that has been returned by the driver at the moment of reservation (see *dma_reserve_channel()*.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed. Expect status message.
RV_MEMORY_ERR	Insufficient memory to create the context.
RV_NOT_READY	The driver is not able to handle this request at this moment. (ENTITY initialization is not done correctly).

Event Return

- **DMA_STATUS_RSP_MSG**

This message is send to the client to return the status and result of the requested action. Other API's also uses this message to return status or to notify clients of asynchronous events.

structure member ***result.status***

For this action, the value of the message structure member ***result.status*** can have the following values:

Message	Definition
DMA_OK	The provided information is validated and

	accepted. The channel reservation request is removed from the queue.
DMA_INVALID_PARAMETER	The parameter is incorrect.
DMA_NOT_READY	The driver is not able to handle this request at this moment. (ENTITY initialization is not done correctly or the channel is already dequeued).
DMA_ACTION_NOT_ALLOWED	There is no queued channel reservation request of this client available.
DMA_MEMORY_ERR	The available memory within the DMA ENTITY is insufficient to process the command.

structure member ***result.channel***

For this action, the value of the message structure member ***result.channel*** holds the provided channel_queue_id.

Note that this API functionality can also be invoked with the message DMA_REMOVE_FROM_QUEUE_REQ_MSG.

Current restriction of use

None.

5.3.3 dma_set_channel_parameters

```
T_RV_RET dma_set_channel_parameters (T_DMA_CHANNEL            channel,
                                    T_DMA_CHANNEL_PARAMETERS *channel_info_p)
```

Description

This function allows the programming of the specific channel parameters.
All operational settings required for executing a DMA transfer, has to be provided within the *channel_info* structure. Examples of parameters are:

- The client can indicate if he wants to be notified when the DMA transfer is completed or not.
- The client can indicate if he wants to enable the transfer immediately or later.
- Software or hardware start-source (which hardware source.
- length, endian, hardware priority, data width, DMA mode etc.

The function must be called after the channel reservation is granted and before the DMA transfer is enabled. The function returns immediately and handles the request asynchronously. The message is then validated and handled. The return path is used to inform the client about the result of the message processing and to inform the client of any asynchronous events.

Parameters

- **channel**

channel contains the number of the reserved channel. This number is returned at the moment the reservation is granted.

- **channel_info_p**

channel_info_p must point to a structure containing all the information required to prepare the DMA transfer.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed. Expect status message.
RV_MEMORY_ERR	Insufficient memory to create the context.
RV_NOT_READY	The driver is not able to handle this request at this moment. (ENTITY initialization is not done correctly).

Event Return

- **DMA_STATUS_RSP_MSG**

This message is send to the client to return the status and result of the requested action. Other API's also uses this message to return status or to notify clients of asynchronous events.

structure member ***result.status***

For this action, the value of the message structure member ***result.status*** can have the following values:

Message	Definition
DMA_OK	The provided information is validated and accepted. Hardware settings are updated accordingly.
DMA_INVALID_PARAMETER	The parameter is incorrect.
DMA_NOT_READY	The driver is not able to handle this request at this moment. (ENTITY initialization is not done correctly or the channel is already dequeued).
DMA_ACTION_NOT_ALLOWED	There is an incorrect sequence of API invocations (command ignored).
DMA_MEMORY_ERR	The available memory within the DMA ENTITY is insufficient to process the command.

structure member ***result.channel***

For this action, the value of the message structure member ***result.channel*** holds the processed channel number.

Note that this API functionality can also be invoked with the message DMA_SET_CHANNEL_PARAMETERS_REQ_MSG.

Current restriction of use

None.

5.3.4 dma_enable_transfer

```
T_RV_RET dma_enable_transfer( T_DMA_CHANNEL channel)
```

Description

This function allows the start of the prepared DMA transfer.If the DMA parameters have been set-up to use the hardware synchronisation, this is armed. The DMA device can initiate the transfer at any time.If the DMA parameters have been set-up to use software synchronisation, the DMA transfer is started immediately.The function can be called again, after a transfer is completed (if the channel is not released). Called more then once before the transfer is completed shall result in the error DMA_CHANNEL_ENABLED.

The function must be called after the channel reservation is granted and channel parameters has been set. The function returns immediately and handles the request asynchronously. The message is then validated and handled. The return path is used to inform the client about the result of the message processing and to inform the client of any asynchronous events.

Parameters

- **channel**

channel contains the number of the reserved channel. This number has been returned at the moment the reservation was granted.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed. Expect status message.
RV_MEMORY_ERR	Insufficient memory to create the context.
RV_NOT_READY	The driver is not able to handle this request at this moment. (ENTITY initialization is not done correctly).

Event Return

- **DMA_STATUS_RSP_MSG**

This message is send to the client to return the status and result of the requested action. Other API's also uses this message to return status or to notify clients of asynchronous events.

structure member ***result.status***

For this action, the value of the message structure member ***result.status*** can have the following values:

Message	Definition
DMA_OK	The provided information is validated and accepted. The DMA is started or armed.
DMA_INVALID_PARAMETER	The parameter is incorrect.
DMA_NOT_READY	The driver is not able to handle this request at this moment.
DMA_ACTION_NOT_ALLOWED	There is an incorrect sequence of API invocations (command ignored).
DMA_MEMORY_ERR	The available memory within the DMA ENTITY

	is insufficient to process the command.
DMA_CHANNEL_ENABLED	The channel is already enabled.

structure member ***result.channel***

For this action, the value of the message structure member ***result.channel*** holds the processed channel number.

Note that this API functionality can also be invoked with the message DMA_ENABLE_TRANSFER_REQ_MSG.

Current restriction of use

None.

5.3.5 dma_release_channel

```
T_RV_RET dma_release_channel (T_DMA_CHANNEL channel)
```

Description

This function allows the release of the reserved DMA channel. Depending on the state of transfer, the release may be immediately or postponed till the busy transfer is completed.
If the release must be postponed, the client shall be notified asynchronous as soon as the release is possible (status message DMA_CHANNEL_RELEASED, using the return path provided at reserving time).

Releasing a channel enables the DMA ENTITY to serve a queued reservation request. The channel shall be assigned to a queued client with the highest priority.
When a client receives an asynchronous error event from the DMA ENTITY, the client has still the obligation to release the channel.

The function returns immediately and handles the request asynchronously. The message is then validated and handled. The return path is used to inform the client about the result of the message processing and to inform the client of any asynchronous events.

Parameters

- **channel**

channel contains the number of the channel to be released. This identification has been returned at the moment the reservation was granted.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed. Expect status message.
RV_MEMORY_ERR	Insufficient memory to create the context.
RV_NOT_READY	The driver is not able to handle this request at this moment. (ENTITY initialization is not done correctly).

Event Return

- **DMA_STATUS_RSP_MSG**

This message is send to the client to return the status and result of the requested action. Other API's also uses this message to return status or to notify clients of asynchronous events.

structure member ***result.status***

For this action, the value of the message structure member ***result.status*** can have the following values:

Message	Definition
DMA_OK	The provided information is validated and accepted. The channel is released.
DMA_INVALID_PARAMETER	The parameter is incorrect.
DMA_NOT_READY	The driver is not able to handle this request at this moment.
DMA_ACTION_NOT_ALLOWED	There is an incorrect sequence of API invocations (command ignored).
DMA_MEMORY_ERR	The available memory within the DMA ENTITY is insufficient to process the command.
DMA_CHANNEL_BUSY	The channel could not be released now. An asynchronous message is send as soon the channel could be released (status DMA_CHANNEL_RELEASED).

structure member ***result.channel***

For this action, the value of the message structure member ***result.channel*** holds the processed channel number.

Optional event return

- **DMA_STATUS_RSP_MSG**

When a channel is released, the DMA driver will check if there are queued reservations left in it's queue buffer. If so, a queued reservation might be processed when it meets the conditions for the freed DMA channel. In that case an additional message is send to the client of the queued reservation. The structure member ***result.status*** holds the value: DMA_QUEUE_PROC.
The structure member ***result.channel*** holds both the processed channel number as well as the queue index provided at the time the reservation was queued.

The information in the structure member ***result.channel*** is stored as follows:
bits 2-0: channel number processed
bits 7-4: provided queue index at the time the reservation was queued.

Note that this API functionality can also be invoked with the message DMA_RELEASE_CHANNEL_REQ_MSG.

Current restriction of use

None.

5.3.6 dma_get_sw_version

```
UINT32 dma_get_sw_version (void)
```

Description

This function returns the driver version.

Parameters

None.

Immediate Return

- **UINT32**

Bit	Name	Function
[0-15]	BUILD	Build number
[16-23]	MINOR	Minor version number
[24-31]	MAJOR	Major version number

Event Return

None.

Current restriction of use

None.

5.4 Message definition

5.4.1 Messages from the driver

5.4.1.1 DMA_STATUS_RSP_MSG

This message is sent to a client to provide the result of a command or to give notice of an asynchronous event occurs.

```
typedef struct {
    T_RV_HDR            hdr;
    T_DMA_RESULT        result;
} T_DMA_STATUS_RSP_MSG;
```

5.4.2 Messages to the driver

5.4.2.1 DMA_RESERVE_CHANNEL_REQ_MSG

This message is send by the client to request a channel.

```
typedef struct {
        T_RV_HDR            hdr;
        T_DMA_SPECIFIC      specific;
        T_DMA_CHANNEL       channel;
        T_DMA_QUEUE         queue;
        T_DMA_SW_PRIORITY       sw_priority;
```

```
                T_RV_RETURN             return_path;
                }T_DMA_RESERVE_CHANNEL_REQ_MSG;
```

5.4.2.2 DMA_REMOVE_FROM_QUEUE_REQ_MSG

This message is send by the client to remove a queued channel reservation.

```
typedef struct {
                T_RV_HDR                hdr;
                T_DMA_CHANNEL       channel_queue_id;
                }T_DMA_REMOVE_FROM_QUEUE_REQ_MSG;
```

5.4.2.3 DMA_SET_CHANNEL_PARAMETERS_REQ_MSG

This message is send by the client to provide detailed channel information specifying the DMA transfer.

```
typedef struct {
                T_RV_HDR                        hdr;
                T_DMA_CHANNEL                   channel;
                T_DMA_CHANNEL_PARAMETERS        channel_info;
                }T_DMA_SET_CHANNEL_PARAMETERS_REQ_MSG;
```

5.4.2.4 DMA_ENABLE_TRANSFER_REQ_MSG

This message is send by the client to enable the DMA transfer.

```
typedef struct {
                T_RV_HDR                hdr;
                T_DMA_CHANNEL           channel;
                }T_DMA_ENABLE_TRANSFER_REQ_MSG;
```

5.4.2.5 DMA_RELEASE_CHANNEL_REQ_MSG

This message is send by the client to release a reserved channel.

```
typedef struct {
                T_RV_HDR                hdr;
                T_DMA_CHANNEL           channel;
                }T_DMA_RELEASE_CHANNEL_REQ_MSG;
```

5.5 SWE State diagram

TBD

5.6 Usage Scenarios

TBD

Chapter 6 EMIF DRIVER

6.1 Introduction

This document outlines

1. Design of EMIF driver for Locosto
2. Configuration information of EMIF chip-selects for specific memory devices connected.
3. Boot time initialization of different chip-selects.

6.2 Locosto EMIF

The External Memory Interface (**EMIF**) is part of MCU sub-system that manages the read/writes between MCU/DMA and external memory (like flashes and SRAMs). The following block diagram shows the various interconnections to EMIF.

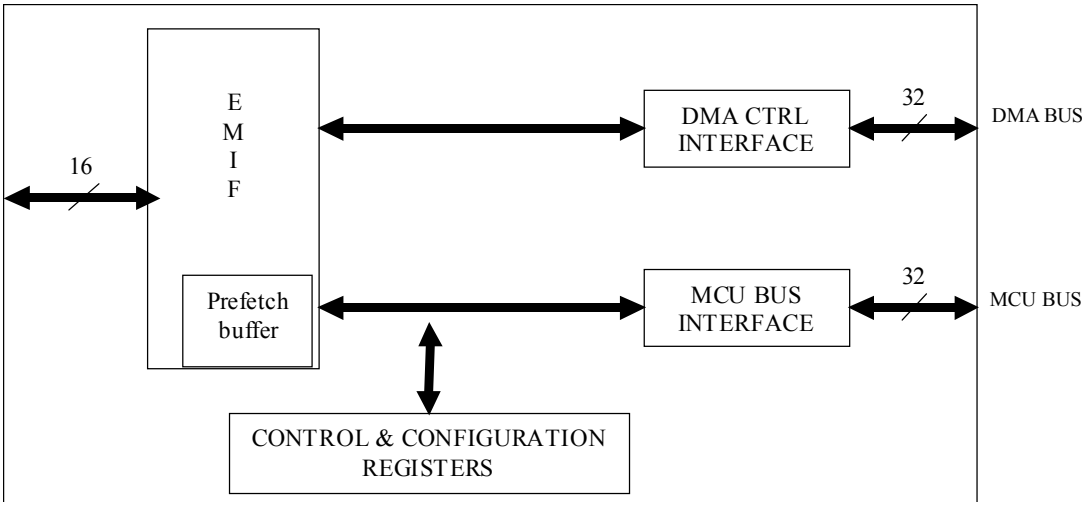


Figure 7 EMIF interface block diagram

Locosto EMIF supports four chip selects (CS0…CS3) for external memories. Each of them has an address range of 32 Mbytes. In Locosto, only two chip selects CS0 and CS3 are used. 28 Mbytes of pSRAM is connected to CS0 and 32 Mbytes of NOR flash is connected to CS3.

6.3 EMIF driver architecture

6.4 EMIF configuration flow

EMIF configuration in Locosto is done in two parts.

1. Default initialization during boot-up
2. Complete configuration

The flow diagram below shows the actual EMIF configuration flow.

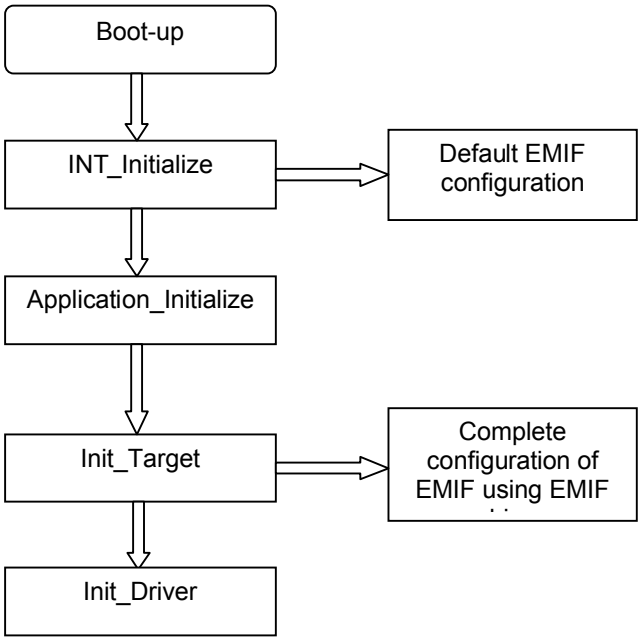


Figure 8 EMIF configuration flow

The default configuration during boot-up is done just to make external memory access work. This doesn't take care of arbitration involved between MCU and DMA and the efficiency of the memory access. This configures the default values for the external memories attached to EMIF. This is done as part of INT_Initialize assembly routine.

The complete configuration of EMIF is done to take care of arbitration involved between MCU and DMA and the efficiency of the external memory access. This is done as part of Init_Target and calls the EMIF driver APIs as described in 6.6 to make the configuration.

6.5 EMIF configuration information

The complete configuration information of different memory attached to EMIF is given below.

6.5.1 CS0 - pSRAM

Memory access mode – Asynchronous mode

No.	Configuration item in EMIF registers	Timing in ns	Cycles for 54Mhz clock
1.	BTWST – Wait states for read to write transition	0	0
2.	WELEN – Write enable length	70	4
3.	WRWST – Wait states for write operation	55	3
4.	RDWST – Wait states for read operation	90	5
5.	OESETUP - cycles inserted from CS low to OE low	40	3
6.	OEHOLD – cycles from OE high to CS high	0	0
7.	ADVHOLD – address valid hold	20	2

6.5.2 CS3 - Flash

Memory access mode – Asynchronous mode

No.	Configuration item in EMIF registers	Timing in ns	Cycles for 54Mhz clock
1.	BTWST – Wait states for read to write transition	0	0
2.	WELEN – Write enable length	50	3
3.	WRWST – Wait states for write operation	80	5
4.	RDWST – Wait states for read operation	70	4
5.	OESETUP - cycles inserted from CS low to OE low	56	4
6.	OEHOLD – cycles from OE high to CS high	0	0
7.	ADVHOLD – address valid hold	12	1

6.6 EMIF driver API

The EMIF driver implements the following API.

6.6.1 f_emif_set_priority

```
SYS_UWORD8 f_emif_set_priority (SYS_UWORD8 d_dma_access,
                                SYS_UWORD8 d_mpu_access)
```

Description

This is to set the number of continuous MCU and DMA accesses.

Parameters

- **d_dma_access**

Number of continuous DMA access.

- **d_mpu_access**

Number of continuous MCU access.

6.6.2 f_emif_set_conf

```
SYS_UWORD8 f__emif_set_conf (T_EMIF_CONF* emif_conf)
```

Description

This API is to do generic EMIF configuration.

Parameters

- **emif_conf**

EMIF configuration as defined below.

typedef struct {

T_EMIF_PREFETCH_MODE d_prefetch_mode; /*Prefetch mode */
T_PDE_STATE d_pde_enable; /*Power down enable*/
T_PWD_STATE d_pwd_enable; /*Global power down enable*/
SYS_UWORD8 flush_prefetch; /* Flush prefetch buffer */
SYS_UWORD8 write_protect; /* write protect control */
} T_EMIF_CONF;

6.6.3 f_emif_cs_mode

```
SYS_UWORD8 f_emif_cs_mode (SYS_UWORD8 d_cs,
                           T_EMIF_CS_CONFIG* p_emif_cs_config)
```

Description

This API is to do the chip-select specific EMIF configuration.

Parameters

- **d_cs**

Chip select

- **p_emif_cs_config**

EMIF chipset specific configuration as defined below.

```
typedef struct {        /* CONF_CSx register configuration */
  SYS_UWORD8  d_wait_read_write_trans;          /* read to write transition wait cycles */
  SYS_UWORD8  d_memmode;                        /* memory mode */
  SYS_UWORD8  d_we;                             /* Write enable cycles */
  SYS_UWORD8  d_wait_write;                     /*Write wait cycles*/
  SYS_UWORD8  d_wait_read;                      /* Read wait cycles*/
  SYS_UWORD8  d_retime;
  SYS_UWORD8  d_flash_clk_div;                  /* Flash clock divider */
  SYS_UWORD8  d_non_full_handshake_mode;
  SYS_UWORD8  d_oe_setup;                       /* Output enable Setup */
  SYS_UWORD8  d_oe_hold;                        /* Output enable hold cycles */
  SYS_UWORD8  d_adv_hold;                       /* Address hold cycles */
  SYS_UWORD8  d_bus_turn_mode;                  /* Bus turn around mode*/
  SYS_UWORD8  d_clk_mask;
  SYS_UWORD8  d_ready_configuration;
 } T_MEMIF_CS_CONFIG;
```

6.6.4 f_emif_abort_conf

```
SYS_UWORD8 f_emif_abort_conf (SYS_UWORD8 d_timeout_enable,
                              SYS_UWORD8 d_timeout);
```

Description

This API does the abort configurations.

Parameters

- **d_timeout_enable**

Enable/Disable abort timeout.

- **d_timeout**

Abort time-out value.

6.6.5 f_emif_abort_status

```
SYS_UWORD8 f_emif_abort_status (T_ABORT_STATUS* p_abort_status);
```

Description

This gets the abort status.

Parameters

- **p_abort_status**

Pointer to T_ABORT_STATUS that returns the abort status of EMIF.

```
typedef struct  {
  SYS_UWORD8 d_abort_state;     /* Current abort state */
  SYS_UWORD8 d_abort_address;   /* Abort address */
  SYS_UWORD8 d_abort_host;      /* abort host MCU or DMA */
```

```
SYS_UWORD8 d_abort_protect;      /* Protect abort */
SYS_UWORD8 d_abort_timeout;      /* Timeout abort*/
} T_ABORT_STATUS;
```

6.6.6 f_emif_protect_conf

```
SYS_UWORD8 f_emif_protect_conf(SYS_UWORD32 d_bound_address,
                               SYS_UWORD32 d_protect_cs,
                               SYS_UWORD32 d_protect_mask);
```

Description

This does the protect configuration.

Parameters

- ***d_bound_address***

Protection bound address

- **d_protect_cs**

Protection chip select

- **d_protect_mask**

Protection mask

6.6.7 f_emif_protect_enable

```
SYS_UWORD8 f_emif_protect_enable();
```

Description

This enables the protection unit of EMIF.

6.6.8 f_emif_protect_conf

```
SYS_UWORD8 f_emif_api_rhea_conf(SYS_UWORD8 strobe0_access_size_adapt,
                                SYS_UWORD8 strobe1_access_size_adapt,
                                SYS_UWORD8 api_access_size_adapt,
                                SYS_UWORD8 debug_enable);
```

Description

This enables RHEA and API access size adaptation.

Parameters

- **strobe0_access_size_adapt**

Enable/Disable strobe 0 access size adaptation

- **strobe1_access_size_adapt**

  Enable/Disable strobe 1 access size adaptation

- **api_access_size_adapt**

  Enable/Disable API access size adaptation

- **debug_enable**

  Enable/Disable ARM Debug.

6.6.9 f_emif_boot_mode_conf

```
SYS_UWORD8 f_emif_boot_mode_conf (SYS_UWORD8 boot_ctrl,
                                  SYS_UWORD8 secure_mem_select);
```

Description

This API does the boot mode configurations.

Parameters

- **boot_ctrl**

Gives whether the boot should be performed from internal memory or external memory

- **secure_mem_select**

Secure memory is MCU ROM/Internal RAM.

6.6.10 f_emif_debug_unit_enable

```
SYS_UWORD8 f_emif_debug_unit_enable (SYS_UWORD8 enable);
```

Description

This enables/disables the debug-unit.

Parameters

- **enable**

Enables or disables debug unit using C_EMIF_ENABLE_DU and C_EMIF_DISABLE_DU

Chapter 7 RFS

7.1 Introduction

This document is the programmer's manual for RFS (Riviera Files System). RFS is a file system with an API inspired by the POSIX file I/O interface. Objects in RFS are hierarchically organized in directories and sub-directories. This chapter describes the RFS in general and the remaining part of the document concerns the RFS API.

This RFS interface intends to provide an easy access to RFS for applications; it gathers the applications requirements for a common File System

7.2 RFS features

The RFS is capable to support following features:

- Well known API (POSIX look-alike)
- The File System Core that actually knows the file structure (FAT/FFS/etc..) on the media is interchangeable.
- Support removable devices (e.g.: MMC, SD…)
- Dynamic object allocation and garbage collection
- Power fail recovery
- Wear leveling
- Device mounting

7.3 Device independent

The application programmer doesn't have to know about the flash, in which the file is stored, because RFS is independent from the underlying flash device hardware.

7.4 Storage Device Auto-Detection

RFS will mount every available device upon boot process and automatically mount removable devices on insertion and unmount them on removal. The information whether a device is inserted or removed is provided by the GBI. When these situations occur, the RFS is responsible for passing the information to the file system cores.

7.5 Mount point

A mount point is a file system object, typically a directory associated with every storage device and its partitions. The mount point is the root directory of any given storage device partition. The root directory of a mount point is the highest directory in the directory hierarchy. This means the RFS doesn't have a root directory indicated by '/'.

During the boot sequence information of all devices is gathered by the GBI plug-ins. The GBI (see [**Error! Reference source not found.**]) is among others responsible for creating the mount point names. Next a principle is described, which could be used. Note, this is just an example, the exact principle used, is the responsibility of the GBI.

When there is only one occurrence of a storage device type and this device has one partition, the mount point will be indicated without a specific identifier. If there is more than one occurrence of a storage device type, they will be distinguished by a letter. If a given storage device has more than one partition, an index will be used to distinguish them. The device name and the number of occurrence and partition information could be separated by a '-'. This decision however can be made during the configuration phase.

Example of mountpount information gathered after the boot sequence:

Storage Device	Mountpoint (root directory)
NOR flash	/nor
NAND flash	/nand

SD flash	/sd
1st MMC 1st partition	/mmc-a1
1st MMC 2nd partition	/mmc-a2
2nd MMC (one partition)	/mmc-b

When a removal storage device is inserted during runtime and there is already a similar device inserted the mount point of the new device will identified with a letter. This letter becomes the successor of the last used identifier letter for this storage device type. When no identifier letter was used before (one device available during the boot sequence), this letter will be 'a'.

For example, when there is initially one MMC storage device inserted, it's mount point name will be '/mmc'. After inserting another MMC storage device, it's mount point name will become '/mmc-a'.

When there is a need to retrieve information about mount points, a sequence of following actions can be performed. First the root directory should be opened with the function rfs_opendir(), whereby the given pathname is '/'. The mount point names in the root directory are read by using the function rfs_readdir(). Reading the mount point directory entries are performed sequential until all entries are read (the function rfs_readdir returns zero). After a mount point name is available the statistics of mount point can be obtained by using the rfs_stat() function, whereby the given pathname is the mount point name. The statistic information contains among others size information of the mount point.

For more information about reading directory entries, see chapter 7.16.8 and for more information about reading directory entries, see chapter 7.16.3.

7.6 Limitations

- RFS object names (files and directories) have a maximum length based on a minimum guaranteed value independent of file system core below (This value will be set in the system constant: RFS_FILENAME_MAX).
- RFS path names have a maximum length based on a minimum guaranteed value independent of file system core below (This value will be set in the system constant: RFS_PATHNAME_MAX).
- RFS has a maximum number of currently opened files within RFS. This value is configurable. The default value is 10.
- RFS has a maximum number of currently opened directories within RFS. This value is configurable. The default value is 10.
- Some of the RFS API functionality (e.g. permission attributes) is not available for particular file systems. In this case the file system ignores the feature and returns ok.

7.7 Performance

Besides depending on the system load, performance is also depending on the hardware and software environment. This environment affects the read and write performance of RFS.

Read performance is almost exclusively affected by the flash memory speed. Write performance can be severely degraded if the file system is near full and fragmented. Otherwise write performance depends on the write speed of the flash device and the size of the buffers being written. In general, larger buffers mean higher throughput.

7.8 Asynchronous and Synchronous I/O operations

The definition of asynchronous and synchronous operations is based on the fact whether operations are running in the context of a client or running in the context of it's own process. An operation is called synchronous or blocking when the operation runs in the context of client. When an operation runs in the RFS its own context, it is a non-blocking or asynchronous operation.

The decision whether the RFS operations should be blocking or non-blocking is based on the fact, whether the operation is time-consuming or not. For a time consuming operation it can be decided to use non-blocking operation.

To obtain non-block operations, there are several mechanisms:

1. Creating asynchronous operations via the POSIX compliant RFS API interfaces.
2. Make use of Riviera compliant non-blocking RFS API functions.
3. The client/caller of blocking RFS API functions, creates its own non-blocking mechanism

The three non-blocking possibilities will be described in detail in following sub-chapters.

7.9 POSIX compliant non-blocking

In order to make calls non-blocking, the POSIX compliant interface rfs_fcntl() supports a possibility to enable or disable asynchronous I/O operations. The rfs_fcntl() has only control over open files.

When there is a need for asynchronous operations for an open file, the programmer should first enable asynchronous I/O operations (by setting F_SETFL to O_AIO). For asynchronous operations, the RFS requires that all data passed to a non-blocking function (file descriptor and file data) must be valid until the operation has finished.

In order to fulfil this requirement, especially in the case of dynamically allocated memory, the application should provide either a callback function pointer or a message, as specified at the time of calling, in order to get a notification when the operation has finished. The concept of specifying a call back function or a message to be sent on completion of a modify operation, is generally called a confirmation path or a return path. When this notification has been received, the application can free the memory.

As mentioned the client/caller can define which return mechanism should be used. For that purpose, it must provide a return_path. The generic return_path type is a C structure, defined as:

```
typedef struct {
                              T_RVF_ADDR_ID   addr_id;
                              VOID   (*callback_func)(void *);
            } T_RV_RETURN;
```

Most of the time , message answers are sent back using the addr_id information. In that case, the callback function pointer is NULL and the addr_id field is set to the addr_id of the ENTITY that gets the answer. This receiving ENTITY is responsible for de-allocating the message buffer.

In some cases, it may be very useful that a ENTITY sends the message answer by using a callback function and not directly by sending message in a mailbox. In this case, the sending ENTITY is responsible for de-allocating the message buffer.

Setting the return path with the POSIX compliant interface the rfs_fcntl(), can be achieve using the flag F_SETRETPATH.

When there is no more need for asynchronous operations for an open file, the programmer should disable asynchronous I/O operations (by setting F_SETFL to O_SIO).

Example of POSIX compliant non-blocking mechanism in case of a write operation:

```
/* Open file */
fd = rfs_open("/tmp/file", O_WRONLY | O_CREAT, S_IWUSR);

/* force asynchronous I/O operations */
error = rfs_fcntl(fd, F_SETFL, O_AIO);

/* specifying a return path, needed for notification and de-allocation of memory */

ret_path.addr_id       = xxx_get_addr_id();
ret_path.callback_func = NULL;

/* Set return path */
error = rfs_fcntl(fd, F_SETRETPATH, ret_path);

/* start operation */
error = rfs_write(fd, buf, &size);

/* Client waits for ready response */
xxx_wait_for_message (XXX_READY_RSP_MSG);

/* go back to default  synchronous I/O operations */
error = rfs_fcntl(fd, F_SETFL, O_SIO);
```

7.10 Riviera compliant non-blocking

In many cases the RFS API operates with object names instead of a file descriptor. In these cases the application should call a non-blocking RFS API function (functions suffix '_nb'). These API functions send a message to the RFS background task. The API function then returns and the caller's task continue to run. This means that the RFS operation has been scheduled and has not begun execution. Later, when the RFS task gets the CPU, it will read the message and execute the requested operation.

When it is finished, it will return the result of the operation to the caller by means of the return path mechanism. This will be either a message or a callback function, as specified at the time of calling. Also all exceptions are returned through this return path mechanism.

For more information about the return path, see chapter 7.9 and for more information about the complete Riviera concepts, see [**Error! Reference source not found.**].

7.11 Non-blocking created by the client

When a blocking RFS API function is used by an application and it is not possible to use the non-blocking variant of that RFS function, the application can create it's own synchronisation mechanism. For example using the mutex/semaphores and sleep principle.

7.12 Request and response pairing

To be able to compare a response of an operation with the request previously made, a principle with returning a pair value in the response message is introduced. The pair value depends on the request made.

When the request concerns an open file operation, the pair value in the response message will consist of the file descriptor, which is handed over by the start of the operation. The client can use the returned file descriptor to pair it with the file descriptor used at the start of the request. When multiple requests are made to an open file, only one request is in progress. The other requests are queued and executed sequential.

In case the request doesn't concern an operation on an open file (operations, which don't have file descriptor as an input parameter), the request (non-blocking) function returns, in case of success, a unique pair ID (a positive value of the type T_RFS_RET). When the requested operation is finished, the pair value in the response message contains the same unique pair ID (also of type T_RFS_RET). This means the unique pair ID should be handed over through the entire function call cycle. When multiple requests of the same type are made, only one request is in progress. The other requests are queued and executed sequential.

The described principles only concern asynchronous functions.

7.13 Permission attributes

Everything the RFS manages with regards to objects has a set of permissions governing who can read, write, and execute the resource.

The POSIX standard defines three possible types of object access actors: the user who owns it (u), another user in the same system group (g), any other user in any other group (o). The permission attributes of a file are defined as execute (x), write (w) or read (r). It is possible to combine these attributes. Not all file system cores support object actors and modes. In this case user will become the default actor.

This overall numerical definition within a RFS is following:

	User			Group			Other	
symbolic	value	mode bit	symbolic	value	mode bit	symbolic	value	mode bit
u+x	0x0100	S_IXUSR	g+x	0x0010	S_IXGRP	o+x	0x0001	S_IXOTH
u+w	0x0200	S_IWUSR	g+w	0x0020	S_IWGRP	o+w	0x0002	S_IWOTH
u+r	0x0400	S_IRUSR	g+r	0x0040	S_IRGRP	o+r	0x0004	S_IROTH
u+rwx	0x0700	S_IRWXU	g+rwx	0x0070	S_IRWXG	o+rwx	0x0007	S_IRWXO

Directories are also treated as files. They have read, write, and execute permissions. The executable bit for a directory has a slightly different meaning than that of files. When a directory is marked executable, it means it can be traversed into. In this case it is possible to change directory into.

In order to perform a directory listing, read and execute permission must be set on the directory, whilst to delete a file that one knows the name of, it is necessary to have write and execute permissions to the directory containing the file.

7.14 Interface description

7.15 POSIX compliant service functions definition

7.15.1 rfs_fcntl

```
T_RFS_RET rfs_fcntl( T_RFS_FD       fd,
                     INT8           cmd,
                     …              /* optional, void* arg */)
```

Description

This function provides control on the properties of a file that is already open. The argument *fd* is a descriptor to be operated on by *cmd* as described below. The third parameter is called *arg* and is technically a pointer to *void*, but the interpretation depends on the command.

This function is a synchronous function. Switching from synchronous to asynchronous operations or vice-versa, by setting flag F_SETFL, has only effect on the succeding operations (like file writing or reading) and not on the function rfs_fcntl() itself. When a switch from asynchronous operations to synchronous operations is made and there are some pending asynchronous operations, the return_path of these pending operations should stay valid.

Parameters

- **fd**

File descriptor obtained when the file was opened.

- **cmd**

The commands are:

id	Definition
F_SETFL	Set the file status associated with the file descriptor fd (arg is interpreted as an UINT8)
F_GETFL	Get the file status associated with fd (*arg* is ignored)
F_SETRETPATH	Sets the return path to be used for notification (arg is interpreted as an pointer to T_RV_RETURN)

The flags F_SETFL and F_GETFL can be as follows:
O_AIO Force time consuming call to operate asynchronously, the caller will be notified either by the return path (callback function or message).
O_SIO Default blocking (i.e: synchronous) I/O operations

- **arg**

Arguments depending on *cmd*, Possible parameters are the flag (enabling and disabling asynchronous operations) and the command for setting the return path.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(positive value)	Value of flags (in case of successful F_GETFL command execution)
RFS_EOK	Ok. (in case of successful command execution others than F_GETFL)
RFS_EBADFD	Invalid file descriptor.
RFS_EINVALID	invalid argument (Invalid command or invalid return_path)

Event Return

No message is returned.

Current restriction of use

None.

7.15.2 rfs_close

```
T_RFS_RET rfs_close(T_RFS_FD  fd)
```

Description

This function closes an open file.

Parameters

- **fd**

File descriptor obtained when the file was opened.

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_EOK	Ok.
RFS_EBADFD	The file argument is not a valid file descriptor.

Event Return

In case of enabled asynchronous operations, T_RFS_READY_RSP_MSG event is returned containing command index: RFS_CLOSE_RSP. Also the file descriptor used for this initiated operation is returned.

Current restriction of use

None.

7.15.3 rfs_write

```
T_RFS_SIZE rfs_write(   T_RFS_FD    fd
                        const void  *buf,
                        T_RFS_SIZE  size)
```

Description

This function writes data to a previously opened file identified by *fd*. It writes *size* bytes of data from the buffer pointed by *buf* at the current position of the file pointer in the file. When the write operation completes, the current position of the file pointer is set to the end of the newly added data.

If the operation succeeds the (positive) number of written bytes is returned. Otherwise an error is returned (negative value).

Parameters

- **fd**

File descriptor obtained when the file was opened.

- **buf**

Pointer to buffer of data to write.

- **size**

Number of bytes to write.

Immediate Return

- **T_RFS_SIZE**

The possible values are:

id	Definition
(Positive value)	Number of bytes actually written.
RFS_EBADFD	The file argument is not a valid file descriptor.
RFS_EACCES	The file is not writable.
RFS_EBADOP	The file is not open for writing.
RFS_EFBIG	An attempt was made to write a file that exceeds the maximum file size.
RFS_ENOSPACE	Out of data space.
RFS_EDEVICE	Device I/O error

Event Return

In case of enabled asynchronous operations, T_RFS_READY_RSP_MSG event is returned containing command index: RFS_WRITE_RSP. Also the file descriptor used for this initiated operation is returned.

Current restriction of use

None.

7.15.4 rfs_read

```
T_RFS_SIZE rfs_read( T_RFS_FD       fd,
                    void            *buf,
                    T_RFS_SIZE      size)
```

Description

This function reads data from the previously opened file identified by *fd.* A maximum of *size* bytes is read from the file into the buffer pointed by *buf*. The buffer memory needs to be allocated by the client. When the read operation completes, the file pointer is advanced to the end of the data read.

If the operation succeeds the (positive) number of bytes read is returned. Otherwise an error is returned (negative value).

Parameters

- **fd**

File descriptor obtained when the file was opened.

- **buf**

Pointer to a buffer where the data will be copied into (The size of this buffer has to be at least *size* bytes).

- **size**

Maximum number of bytes to read.

Immediate Return

- **T_RFS_SIZE**

The possible values are:

id	Definition
(Positive value)	Number of bytes actually read.
RFS_EBADFD	The file argument is not a valid file descriptor.
RFS_EACCES	The file is not readable.
RFS_EBADOP	The file is not open for reading.
RFS_ENOSPACE	Out of data space.
RFS_EDEVICE	Device I/O error

Event Return

In case of enabled asynchronous operations, T_RFS_READY_RSP_MSG event is returned containing command index: RFS_READ_RSP. Also the file descriptor used for this initiated operation is returned.

Current restriction of use

None.

7.15.5 rfs_lseek

```
T_RFS_OFFSET rfs_lseek( T_RFS_FD          fd,
                        T_RFS_OFFSET      offset,
                        INT8              whence)
```

Description

This function repositions the offset of the file descriptor *fd* to the argument *offset* according to the directive *whence*. The argument *fd* must be an open file descriptor.

If the operation succeeds the (positive) new position of the file pointer is returned. Otherwise an error is returned (negative value).

Note:
When the *whence* is set to RFS_SEEK_END the file offset is set to size of the file plus the offset. In this case new blocks will be added to the file and this can take additional time. This means, also depending on system load, the blocking version (synchronous operation) this function can block the caller for a time.

Parameters

- **fd**

File descriptor obtained when the file was opened.

- **offset**

Offset (in bytes) to move the file pointer.

- **whence**

Reference used to reposition the file pointer defined as follow:

id	Definition
RFS_SEEK_SET	Absolute offset from start of file
RFS_SEEK_CUR	the offset is set to its current location plus *offset* bytes
RFS_SEEK_END	the offset is set to the size of the file plus *offset* bytes.

Immediate Return

- **T_RFS_OFFSET**

The possible values are:

id	Definition
(Positive value)	New position of the file pointer
RFS_EBADFD	The fd argument is not a valid file descriptor.
RFS_EINVALID	The whence argument is not a proper value, or the resulting file offset would be invalid.
RFS_EBADOP	Bad operation. Seek not allowed with the flags used to open the file.
RFS_EDEVICE	Device I/O error

Event Return

In case of enabled asynchronous operations, T_RFS_READY_RSP_MSG event is returned containing command index: RFS_LSEEK_RSP. Also the file descriptor used for this initiated operation is returned.

Current restriction of use

None.

7.15.6 rfs_fchmod

```
T_RFS_RET rfs_fchmod(T_RFS_FD       fd,
                     T_RFS_MODE     mode)
```

Description

The function rfs_fchmod() sets the permission bits of the specified file descriptor *fd* to required *mode*.

A mode is created from *OR'd* permission bit masks defined in chapter 7.18.25.

Parameters

- **fd**

File descriptor obtained when the file was opened.

- **mode**

Specifies the attribute (permission bits) of the file (See chapter 7.18.25).

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_E0K	Ok
RFS_EBADFD	The fd argument is not a valid file descriptor.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_ENOTDIR	A component of the path prefix is not a directory.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EBADNAME	Object's name contains illegal characters.
RFS_ENOTAFILE	Object is not a file.
RFS_ENOENT	No such file or directory.
RFS_EINVALID	Bad mod option
RFS_ELOCKED	The file is locked (already opened in a conflicting mode).

Event Return

In case of enabled asynchronous operations, T_RFS_READY_RSP_MSG event is returned containing command index: RFS_FCHMOD_RSP. Also the file descriptor used for this initiated operation is returned.

Current restriction of use

Some file system types may not support one or more of the mode attribute bits. (They return doing nothing, without an error).

7.15.7 rfs_fstat

```
T_RFS_RET rfs_fstat( T_RFS_FD       fd,
                     T_RFS_STAT     *stat)
```

Description

The rfs_fstat() function obtains information about an open file associated by the file descriptor *fd*. For more details about the returned information concerning the file, see 7.18.29).

The *stat* memory needs to be allocated by the client.

Parameters

- **fd**

File descriptor obtained when the file was opened.

- **stat**

Contains information (meta-data) about the specified object.

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_EOK	Ok.
RFS_ENOENT	Object not found.
RFS_ENOTDIR	A component of the path prefix is not a directory.
RFS_EBADFD	Bad file descriptor.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Object's name contains illegal characters.

Event Return

In case of enabled asynchronous operations, T_RFS_READY_RSP_MSG event is returned containing command index: RFS_FSTAT_RSP. Also the file descriptor used for this initiated operation is returned.

Current restriction of use

Some file system types may not support some specific stat items.

7.15.8 rfs_fsync

```
T_RFS_RET rfs_fsync( T_RFS_FD  fd)
```

Description

This function cause the transfer of all modified data and attributes of *fd*, which wasn't immediately written to the storage device, to the permanent storage device associated with the file described by *fd*. The reason that sometimes the data is not immediately written on the storage device, can be caused by different reasons, like different buffer sizes that are used by various software and hardware components, task scheduling or hardware delays.

To ensure that the data is consistent and really written on the physical media, this function can be used. Saving unwritten data on storage devices is also known as flushing.

Parameters

- **fd**

File descriptor obtained when the file was opened.

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_EOK	Ok.
RFS_EBADFD	Invalid file descriptor.
RFS_ENOSPACE	Out of data space.
RFS_EFSFULL	File system full, no free inodes.
RFS_EDEVICE	Device I/O error

Event Return

In case of enabled asynchronous operations, T_RFS_READY_RSP_MSG event is returned containing command index: RFS_FSYNC_RSP. Also the file descriptor used for this initiated operation is returned.

Current restriction of use

None.

7.16 POSIX and REMU compliant service functions definition

7.16.1 Open file

7.16.1.1 rfs_open

```
T_RFS_FD rfs_open    (  const char        *pathname,
                        T_RFS_FLAGS       flags,
                        T_RFS_MODE        mode)
```

Description

The file specified by *pathname* is opened for reading and/or writing as specified by the argument *flags* and the file descriptor is returned to the calling process. The *flags* argument may indicate the file is to be created if it does not exist (by specifying the RFS_O_CREAT flag), in which case the file is created with mode *mode*. Else the *mode* will be ignored.

If the operation succeeds the file pointer used to mark the current position within the file is set to the beginning of the file and a (positive) file descriptor is returned. Otherwise an error is returned (negative value).

Parameters

- **pathname**

Null terminated string containing the unique name of the file to open or create.

- **flags**

Specifies the attribute used to open the file (see chapter 7.18.24).

- **mode**

Specifies the mode argument (permission bits of the file) and will be used when the RFS_O_CREAT flag is specified in flags (see chapter 7.18.25).

Immediate Return

- **T_RFS_FD**

The possible values are:

id	Definition
(Positive value)	File descriptor of file opened.
RFS_EEXISTS	An object of the same name already exists.
RFS_EACCES	- Search permission is denied for a component of the path prefix. - or the required permissions (for reading and/or writing) are denied for the given flags. - or RFS_O_CREAT is specified, the file does not exist, and the directory in which it is to be created does not permit writing.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Object's name contains illegal characters.
RFS_ENUMFD	Max number of used file descriptors reach
RFS_ENOENT	No such file or directory.
RFS_ENOTDIR	A component of the path prefix is not a directory.
RFS_EINVALID	Bad open flag options.
RFS_ELOCKED	The file is locked (already opened for writing, in a conflicting mode).
RFS_EMOUNT	Invalid mount point

Event Return

No message is returned.

Current restriction of use

None.

7.16.1.2 rfs_open_nb

```
T_RFS_RET rfs_open_nb(  const char        *pathname,
                        T_RFS_FLAGS       flags,
                        T_RFS_MODE        mode,
                        T_RV_RETURN       return_path)
```

Description

This is a non-blocking function variant of rfs_open(). For a detailed description see according chapter.

Parameters

- **pathname**

Null terminated string containing the unique name of the file to open or create.

- **flags**

Specifies the attribute used to open the file (see chapter 7.18.24).

- **mode**

Specifies the mode argument (permission bits of the file) and will be used when the RFS_O_CREAT flag is specified in flags (see chapter 7.18.25).

- **return_path**

Return path for notifications.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(Positive value)	Unique pair ID used to pair a initiated request with a received response.
RFS_EACCES	The RFS is not able to handle this request at this moment.
RFS_EMEMORY	Insufficient memory to create message request
RFS_EINVALID	Invalid argument (return_path is invalid)

Event Return

T_RFS_READY_RSP_MSG event is returned containing command index: RFS_OPEN_RSP. Also a unique pair ID is returned. This pair ID can be used to pair a response with an initiated request.

Current restriction of use

None.

7.16.2 Change mode

7.16.2.1 rfs_chmod

```
T_RFS_RET rfs_chmod( const char     *pathname,
                     T_RFS_MODE     mode)
```

Description

The function rfs_chmod() sets the file permission bits of the file specified by the pathname *pathname* to required mode.

A mode is created from *OR'd* permission bit masks defined in chapter 7.18.25.

Parameters

- **pathname**

Null terminated string containing the unique name of the file for changing the mode.

- **mode**

Specifies the attribute (permission bits) of the file (See chapter 7.18.25).

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_E0K	Ok
RFS_EBADFD	The fd argument is not a valid file descriptor.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_ENOTDIR	A component of the path prefix is not a directory.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EBADNAME	Object's name contains illegal characters.
RFS_ENOTAFILE	Object is not a file.
RFS_ENOENT	No such file or directory.
RFS_EINVALID	Bad mod option
RFS_ELOCKED	The file is locked (already opened in a conflicting mode).
RFS_EMOUNT	Invalid mount point

Event Return

No message is returned.

Current restriction of use

Some file system types may not support one or more of the mode attribute bits. (They return doing nothing, without an error).

7.16.2.2 rfs_chmod_nb

```
T_RFS_RET rfs_chmod_nb( const char        *pathname,
                        T_RFS_MODE        mode,
                        T_RV_RETURN       return_path)
```

Description

This is a non-blocking function variant of rfs_chmod(). For a detailed description see according chapter.

Parameters

- **pathname**

Null terminated string containing the unique name of the file for changing the mode.

- **mode**

Specifies the attribute (permission bits) of the file (See chapter 7.18.25).

- **return_path**

Return path for notifications.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(Positive value)	Unique pair ID used to pair a initiated request with a received response.
RFS_EACCES	The RFS is not able to handle this request at this moment.
RFS_EMEMORY	Insufficient memory to create message request
RFS_EINVALID	Invalid argument (return_path is invalid)

Event Return

T_RFS_READY_RSP_MSG event is returned containing command index: RFS_CHMOD_RSP. Also a unique pair ID is returned. This pair ID can be used to pair a response with an initiated request.

Current restriction of use

Some file system types may not support one or more of the mode attribute bits. (They return doing nothing, without an error).

7.16.3 Retrieve data

7.16.3.1 rfs_stat, rfs_lstat

```
T_RFS_RET rfs_stat(  const char     *pathname,
                     T_RFS_STAT     *stat)
```

Description

The rfs_stat() function obtains information about the file or device associated to the mountpoint pointed to by *pathname*.

If *pathname* is NULL, general information about the file system is returned (e.g. file system limits). If *pathname* is '/mountpoint', information on the device associated to the mount point is returned (e.g. speed data of the mount pount). if *pathname* ends with a directory or file, the appropriate information for the directory or file is returned. For more details about the returned information concerning the pathname, see 7.18.29).

The *stat* memory needs to be allocated by the client.

Parameters

- **stat**

Contains information (meta-data) about the specified object.

- **pathname**

Terminated string containing NULL or the name of the mount pount, file or directory.

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_EOK	Ok.
RFS_ENOENT	Object not found.
RFS_ENOTDIR	A component of the path prefix is not a directory.
RFS_EBADFD	Bad file descriptor.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EMOUNT	Invalid mount point
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Object's name contains illegal characters.

Event Return

No message is returned.

Current restriction of use

Some file system types may not support some specific stat items.

7.16.3.2 rfs_stat_nb, rfs_lstat_nb

```
T_RFS_RET rfs_stat_nb(  const char        *pathname,
                        T_RFS_STAT        *stat,
                        T_RV_RETURN       return_path)
```

Description

This the non-blocking function variant of rfs_stat(). For a detailed description see according chapter.

Parameters

- **pathname**

Null terminated string containing the name of the file or directory.

- **stat**

Contains information (meta-data) about the specified object.

- **return_path**

Return path for notifications.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(Positive value)	Unique pair ID used to pair a initiated request with a received response.
RFS_EACCES	The RFS is not able to handle this request at this moment.
RFS_EMEMORY	Insufficient memory to create message request
RFS_EINVALID	Invalid argument (return_path is invalid)

Event Return

T_RFS_READY_RSP_MSG event is returned containing command index: RFS_STAT_RSP. Also a unique pair ID is returned. This pair ID can be used to pair a response with an initiated request.

Current restriction of use

Some file system types may not support some specific stat items.

7.16.4 Remove

7.16.4.1 rfs_remove

```
T_RFS_RET rfs_remove(const char     *pathname)
```

Description

This function removes the object with the pathname given by pathname. The pathname is a null terminated string. If the object does not exist, RFS_ENOENT is returned. If a directory is to be removed, it must be empty, otherwise RFS_ENOTEMPTY is returned. It is not possible to remove a file that is open.

Parameters

- **pathname**

Null terminated string containing the unique name of the object.

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_EOK	Ok.
RFS_ENOENT	File was not found.
RFS_ENOTDIR	A component of the path is not a directory.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Object's name contains illegal characters.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EACCES	File could not be removed (read-only).
RFS_ELOCKED	The file is open
RFS_EDEVICE	Device I/O error
RFS_EMOUNT	Invalid mount point
RFS_ENOTEMPTY	The named directory contains files other than '.' and '..' in it.

Event Return

No message is returned.

Current restriction of use

None.

7.16.4.2 rfs_remove_nb

```
T_RFS_RET rfs_remove_nb(const char        *pathname,
                        T_RV_RETURN       return_path)
```

Description

This is a non-blocking function variant of rfs_remove(). For a detailed description see according chapter.

Parameters

- **pathname**

Null terminated string containing the unique name of the object.

- **return_path**

Return path for notifications.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(Positive value)	Unique pair ID used to pair a initiated request with a received response.
RFS_EACCES	The RFS is not able to handle this request at this moment.
RFS_EMEMORY	Insufficient memory to create message request
RFS_EINVALID	Invalid argument (return_path is invalid)

Event Return

T_RFS_READY_RSP_MSG event is returned containing command index: RFS_REMOVE_RSP. Also a unique pair ID is returned. This pair ID can be used to pair a response with an initiated request.

Current restriction of use

None.

7.16.5 Rename

7.16.5.1 rfs_rename

```
T_RFS_RET rfs_rename(const char     *oldname,
                    const char     *newname)
```

Description

This function renames files and directories. The names are the full path to the object. It is possible to move the object to a different path simple by specifying a new path in the *newname* string. The *oldname* object must exist and the *newname* must not exist or else an error will be returned.

Renaming (moving) of a file is only granted on the same mountpoint (media partition). If the new path indicates another mountpoint an error is returned.

Parameters

- **oldname**

Null terminated string containing the unique name including the path of the existing object in the File System.

- **newname**

Null terminated string containing the unique name including the path, which *oldname is* desired to change name or location to.

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_EOK	Ok.
RFS_ENOENT	Oldname object does not exist.
RFS_EEXISTS	Newname object already exists.
RFS_EACCES	Object could not be modified (read-only).
RFS_ENOTDIR	A component of the path is not a directory.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Object's name contains illegal characters.
RFS_EFSFULL	Failed to allocate an inode for the changed object.
RFS_ENOTALLOWED	Renaming is not allowed (another new path contains another mountpoint)

Event Return

No message is returned.

Current restriction of use

None.

7.16.5.2 rfs_rename_nb

```
T_RFS_RET rfs_rename_nb(const char        *oldname,
                       const char        *newname,
                       T_RV_RETURN       return_path)
```

Description

This is a non-blocking function variant of rfs_rename(). For a detailed description see according chapter.

Parameters

- **oldname**

Null terminated string containing the unique name including the path of the existing object in the File System.

- **newname**

Null terminated string containing the unique name including the path, which *oldname is* desired to change name or location to.

- **return_path**

Return path for notifications.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(Positive value)	Unique pair ID used to pair a initiated request with a received response.
RFS_EACCES	The RFS is not able to handle this request at this moment.
RFS_EMEMORY	Insufficient memory to create message request
RFS_EINVALID	Invalid argument (return_path is invalid)

Event Return

T_RFS_READY_RSP_MSG event is returned containing command index: RFS_RENAME_RSP. Also a unique pair ID is returned. This pair ID can be used to pair a response with an initiated request.

Current restriction of use

None.

7.16.6 Make directory

7.16.6.1 rfs_mkdir

```
T_RFS_RET rfs_mkdir( const char     *pathname,
                     T_RFS_MODE     mode)
```

Description

This function creates a directory with the pathname given by *pathname*. The pathname is a null terminated string. All components of the pathname must be already existing directories. This means that it is not possible to rfs_mkdir("/gsm/rf/tx") if the directories /gsm/ and /gsm/rf are not already created.

Parameters

- **pathname**

Null terminated string containing the unique name of the directory to create

- **mode**

Specifies the attribute (permission bits) of the directory (See chapter 7.18.25).

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_EOK	Ok.
RFS_EEXISTS	Directory already exists.
RFS_ENOTDIR	A component of the path is not a directory.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EBADNAME	Name of the directory contains illegal characters
RFS_ENOSPACE	Failed to allocate space for object's data.
RFS_EFSFULL	Failed to allocate an inode for the object.
RFS_EDEVICE	Device I/O error
RFS_EMOUNT	Invalid mount point

Event Return

No message is returned.

Current restriction of use

None.

7.16.6.2 rfs_mkdir_nb

```
T_RFS_RET rfs_mkdir_nb( const char        *pathname,
                        T_RFS_MODE        mode,
                        T_RV_RETURN       return_path)
```

Description

This is a non-blocking function variant of rfs_mkdir(). For a detailed description see according chapter.

Parameters

- **pathname**

Null terminated string containing the unique name of the directory to create

- **mode**

Specifies the attribute (permission bits) of the directory (See chapter 7.18.25).

- **return_path**

Return path for notifications.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(Positive value)	Unique pair ID used to pair a initiated request with a received response.
RFS_EACCES	The RFS is not able to handle this request at this moment.
RFS_EMEMORY	Insufficient memory to create message request
RFS_EINVALID	Invalid argument (return_path is invalid)

Event Return

T_RFS_READY_RSP_MSG event is returned containing command index: RFS_MKDIR_RSP. Also a unique pair ID is returned. This pair ID can be used to pair a response with an initiated request.

Current restriction of use

None.

7.16.7 Remove directory

7.16.7.1 rfs_rmdir

```
T_RFS_RET rfs_rmdir( const char     *pathname)
```

Description

This function removes a directory file whose name is given by *pathname*. The directory must be empty.

Parameters

- **pathname**

Null terminated string containing the unique name of the directory to remove.

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_EOK	Ok.
RFS_ENOENT	The named directory does not exist
RFS_ENOTDIR	A component of the path is not a directory.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_ENOTEMPTY	The named directory contains files other than '.' and '..' in it.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EACCES	No write permission to delete the directory entry
RFS_EBUSY	The directory to be removed is the mount point for a mounted file system or the current directory.
RFS_EBADNAME	Name of the directory contains illegal characters
RFS_EMOUNT	Invalid mount point

Event Return

No message is returned.

Current restriction of use

None.

7.16.7.2 rfs_rmdir_nb

```
T_RFS_RET rfs_rmdir_nb( const char        *pathname,
                        T_RV_RETURN       return_path)
```

Description

This is a non-blocking function variant of rfs_rmdir(). For a detailed description see according chapter.

Parameters

- **pathname**

Null terminated string containing the unique name of the directory to remove.

- **return_path**

Return path for notifications.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(Positive value)	Unique pair ID used to pair a initiated request with a received response.
RFS_EACCES	The RFS is not able to handle this request at this moment.
RFS_EMEMORY	Insufficient memory to create message request
RFS_EINVALID	Invalid argument (return_path is invalid)

Event Return

T_RFS_READY_RSP_MSG event is returned containing command index: RFS_RMDIR_RSP. Also a unique pair ID is returned. This pair ID can be used to pair a response with an initiated request.

Current restriction of use

None.

7.16.8 Directory contents determination

7.16.8.1 Introduction

The RFS knows two basic directory operations: open directory and reading directory, to determine the names of files and subdirectories in a directory.

A directory is opened for reading by calling the open directory function and specifying the name of the directory to be opened. The function call returns a pointer to a directory descriptor, which identifies a directory stream. The stream is initially positioned at the first entry in the directory.

Once a directory stream is opened, the reading directory function is used to obtain individual entries from it. Each call to this function returns information of one directory entry, in sequence from the start of the directory. The reading directory function returns the name of the file (or directory) and its name size. In order to read all entries in a directory, the reading directory function should be called until it returns zero.

7.16.8.2 Open directory

7.16.8.2.1 rfs_opendir

```
T_RFS_SIZE *rfs_opendir(const char  *pathname,
                        T_RFS_DIR   *dirp)
```

Description

This function opens the directory named by *pathname* and associates a directory stream with it pointed by pointer *dirp*. This pointer can be used to identify the directory stream in subsequent operations.

If the operation succeeds, the function returns the number of objects in the directory. Otherwise an error is returned (negative value).

Parameters

- **pathname**

Null terminated string containing the unique name of the directory we want to open.

- **dirp**

Pointer to a RFS structure, which has to be used for the read directory operations. The user is responsible to allocate this structure.

Immediate Return

- **T_RFS_SIZE**

The possible values are:

id	Definition
(Positive value)	Number of objects in the specified directory (At the time of call).
RFS_ENOTDIR	A component of the path is not a directory.
RFS_ENOENT	Directory not found.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Name of the directory contains illegal characters
RFS_ENOSPACE	Out of data space.

RFS_EMOUNT	Invalid mount point

Event Return

No message is returned.

Current restriction of use

None.

7.16.8.2.2 rfs_opendir_nb

```
T_RFS_RET rfs_opendir_nb(const char       *pathname,
                         T_RFS_DIR        *dirp,
                         T_RV_RETURN      return_path)
```

Description

This is a non-blocking function variant of rfs_opendir(). For a detailed description see according chapter.

Parameters

- **pathname**

Null terminated string containing the unique name of the directory we want to open.

- **dirp**

Pointer to a RFS structure, which has to be used for the read directory operations. The user is responsible to allocate this structure.

- **return_path**

Return path for notifications.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(Positive value)	Unique pair ID used to pair a initiated request with a received response.
RFS_EACCES	The RFS is not able to handle this request at this moment.
RFS_EMEMORY	Insufficient memory to create message request
RFS_EINVALID	Invalid argument (return_path is invalid)

Event Return

T_RFS_READY_RSP_MSG event is returned containing command index: RFS_OPENDIR_RSP. Also a unique pair ID is returned. This pair ID can be used to pair a response with an initiated request.

Current restriction of use

None.

7.16.8.3 Read directory

7.16.8.3.1 rfs_readdir

```
T_RFS_SIZE rfs_readdir( T_RFS_DIR         *dirp,
                        char              *buf,
                        T_RFS_SIZE        size)
```

Description

This function reads an entry from a directory previously opened by rfs_opendir(). The input parameter is a pointer to the opened directory stream (specified by the argument *dirp*). The structure associated with the pointer *dirp,* keeps track of the directory entry last read by the function rfs_readdir(). To achieve this, the pointer *dirp* is also an output parameter.

The other (output) parameters concern directory entry information, which is read. This information exists of a pointer to a buffer containing the name of the entry and the size of the buffer. The buffer memory needs to be allocated by the client.

A positive return value denotes that the buffer pointed to by buf, contains the null-terminated name of the entry found. A zero is returned if there were no more entries in the directory and the buffer pointed to by buf is left untouched. The function returns a negative value if an exception is encountered.

In order to read all entries in a directory, rfs_readdir() should be called until it returns zero.

Parameters

- **dirp**

(1) Pointer to a T_RFS_DIR structure obtained in a previous call to rfs_opendir(), representing a directory stream.

- **buf**

Pointer to a buffer, which should contain the name of the directory entry.

- **size**

Size in bytes of the buffer pointed by *buf.*

Immediate Return

- **T_RFS_SIZE**

The possible values are:

id	Definition
(Positive value)	Number of bytes actually read.
RFS_EBADDIR	Invalid directory descriptor
RFS_ENOSPACE	Out of data space.

Event Return

No message is returned.

Current restriction of use

None.

7.16.8.3.2 rfs_readdir_nb

```
T_RFS_RET rfs_readdir_nb(T_RFS_DIR        *dirp,
                         char             *buf,
                         T_RFS_SIZE       size,
                         T_RV_RETURN      return_path)
```

Description

This is a non-blocking function variant of rfs_readdir(). For a detailed description see according chapter.

Parameters

- **dirp**

(2) Pointer to a T_RFS_DIR structure obtained in a previous call to rfs_opendir() or rfs_opendir_nb(), representing a directory stream.

- **buf**

Pointer to a buffer, which should contain the name of the directory entry.

- **size**

Size in bytes of the buffer pointed by *buf.*

- **return_path**

Return path for notifications.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(Positive value)	Unique pair ID used to pair a initiated request with a received response.
RFS_EACCES	The RFS is not able to handle this request at this moment.
RFS_EMEMORY	Insufficient memory to create message request
RFS_EINVALID	Invalid argument (return_path is invalid)

Event Return

T_RFS_READY_RSP_MSG event is returned containing command index: RFS_READDIR_RSP. Also a unique pair ID is returned. This pair ID can be used to pair a response with an initiated request.

Current restriction of use

None.

7.16.9 Pre-formatting

7.16.9.1 rfs_preformat

```
T_RFS_RET rfs_preformat (const char *pathname,
                         UINT16     magic)
```

Description

With this pre-format function it is only possible to erase the data in a media partition. Within RFS a media partition is the same as a mountpoint. For this pre-format function it is required that the given *pathname* should be a mountpunt '/mountpoint'. If this is not the case, no pre-formatting takes place and an error is returned.

The pre-format operation cannot be reversed or undone. Note that depending on the underlying flash hardware, the pre-format operation can take anything from a few milliseconds to several seconds. Most flash memories in a normal environment take around one second (typical) to erase each sector. The magic number must equal the hexadecimal constant 0xDEAD. If the magic number given by magic is incorrect, RFS_EMAGIC is returned.

Parameters

- **pathname**

Null terminated string containing the name of the partition to format.

- **magic**

Magic value to access the function. Must be 0xDEAD.

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_EOK	Ok.
RFS_EMAGIC	Magic number is incorrect.
RFS_EINVALID	An erase operation is currently in progress. Retry the operation again later.
RFS_ENODEVICE	The flash device is unknown (not supported).
RFS_EMEMORY	Message allocation failed.
RFS_EMSGSEND	Message sending failed.
RFS_EDEVICE	Device I/O error
RFS_ENOTALLOWED	Pre-formatting is not allowed (pathname is no mountpoint)

Event Return

No message is returned.

Current restriction of use

None.

7.16.9.2 rfs_preformat_nb

```
T_RFS_RET rfs_preformat_nb(const char     *pathname,
                           UINT16         magic,
                           T_RV_RETURN    return_path)
```

Description

This is a non-blocking function variant of rfs_preformat(). For a detailed description see according chapter.

Parameters

- **pathname**

Null terminated string containing the name of the partition to format.

- **magic**

Magic value to access the function. Must be 0xDEAD.

- **return_path**

Return path for notifications.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(Positive value)	Unique pair ID used to pair a initiated request with a received response.
RFS_EACCES	The RFS is not able to handle this request at this moment.
RFS_EMEMORY	Insufficient memory to create message request
RFS_EINVALID	Invalid argument (return_path is invalid)

Event Return

T_RFS_READY_RSP_MSG event is returned containing command index: RFS_PREFORMAT_RSP. Also a unique pair ID is returned. This pair ID can be used to pair a response with an initiated request.

Current restriction of use

None.

7.16.10 Formatting

7.16.10.1 rfs_format

```
T_RFS_RET rfs_format(const char     *pathname,
                     const char     *name,
                     UINT16         magic)
```

Description

This function formats the RFS. With this format function a new file system is created on the requested media partition. Within RFS, a media partition is the same as a mount point. For this format function it is required that the given *pathname* should be a mount punt '/mountpoint'. If this is not the case, no formatting takes place and an error is returned.

With the optional name given by the argument *name* the volume name of the root directory. However the name is completely ignored and has absolutely no meaning to RFS. It can not be read or retrieved later on. If *name* is the null pointer, a default name is used. Otherwise *name* is a null terminated string and optionally followed by some arbitrary descriptive name for the RFS volume.

This function must be called after rfs_preformat() and before any other operation on RFS. In order to avoid spurious calls of this dangerous, unrecoverable function, the *magic* number must have the hexadecimal value of 0x2BAD to format the flash.

Parameters

- **pathname**

Null terminated string containing the name of the partition to format.

- **name**

Null terminated string containing the name of the RFS volume.

- **magic**

Magic value to access the function. Must be 0x2BAD.

Immediate Return

- **T_RFS_RET**

The possible values are:

id	Definition
RFS_EOK	Ok.
RFS_EAGAIN	Previous RFS_preformat() has not finished yet.
RFS_EINVALID	Magic number is incorrect.
RFS_EBADNAME	Name contains illegal characters
RFS_EMEMORY	Message allocation failed.
RFS_EMSGSEND	Message sending failed.
RFS_EDEVICE	Device I/O error
RFS_ENOTALLOWED	Formatting is not allowed (pathname is no mountpoint)

Event Return

No message is returned.

Current restriction of use

None.

7.16.10.2 rfs_format_nb

```
T_RFS_RET rfs_format_nb(const char        *pathname,
                        const char        *name,
                        UINT16            magic,
                        T_RV_RETURN       return_path)
```

Description

This is a non-blocking function variant of rfs_format(). For a detailed description see according chapter.

Parameters

- **pathname**

Null terminated string containing the name of the partition to format.

- **name**

Null terminated string containing the name of the RFS volume.

- **magic**

Magic value to access the function. Must be 0x2BAD.

- **return_path**

Return path for notifications.

Immediate Return

- **T_RFS_RET**

The possible values are:

Id	Definition
(Positive value)	Unique pair ID used to pair a initiated request with a received response.
RFS_EACCES	The RFS is not able to handle this request at this moment.
RFS_EMEMORY	Insufficient memory to create message request
RFS_EINVALID	Invalid argument (return_path is invalid)

Event Return

T_RFS_READY_RSP_MSG event is returned containing command index: RFS_FORMAT_RSP. Also a unique pair ID is returned. This pair ID can be used to pair a response with an initiated request.

Current restriction of use

None.

7.17 Make symbolic link

7.17.1 rfs_symlink

```
T_RFS_RET rfs_symlink ( const char  *name1, const char*name2)
```

Description

This function creates a symbolic link. The symbolic link name2 is created to name1 (name2 is the name of the symbolic link created, name1 is the string used in creating the symbolic link).
The name may be an arbitrary path name. The files need not be on the same file system and also the file specified by name1 needs not to exist at all.

To create a directory link both name1 and name2 character string must end with a '/' character.

Parameters

- **name1**

Null terminated string containing the unique name of the symbolic link to be created.

- **name2**

Null terminated string containing the name of the actual object.

Immediate Return

T_RFS_RET

7.18 Message definition

7.18.1 Close

7.18.1.1 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_CLOSE_RSP. Also the file descriptor used for this initiated operation is returned.

The possible values for 'result' are:

id	Definition
RFS_EOK	Ok.
RFS_EBADFD	The file argument is not a valid file descriptor.

7.18.2 Write

7.18.2.1 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_WRITE_RSP. Also the file descriptor used for this initiated operation is returned.

The possible values for 'result' are:

id	Definition
(Positive value)	Number of bytes actually written.
RFS_EBADFD	The file argument is not a valid file descriptor.
RFS_EACCES	The file is not writable.
RFS_EBADOP	The file is not open for writing.
RFS_EFBIG	An attempt was made to write a file that exceeds the maximum file size.

RFS_ENOSPACE	Out of data space.
RFS_EDEVICE	Device I/O error

7.18.3 Read

7.18.3.1 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_READ_RSP. Also the file descriptor used for this initiated operation is returned.

The possible values for 'result' are:

id	Definition
(Positive value)	Number of bytes actually read.
RFS_EBADFD	The file argument is not a valid file descriptor.
RFS_EACCES	The file is not readable.
RFS_EBADOP	The file is not open for reading.
RFS_ENOSPACE	Out of data space.
RFS_EDEVICE	Device I/O error

7.18.4 Lseek

7.18.4.1 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_LSEEK_RSP. Also the file descriptor used for this initiated operation is returned.
The possible values for 'result' are:

id	Definition
(Positive value)	New position of the file pointer
RFS_EBADFD	The fd argument is not a valid file descriptor.
RFS_EINVALID	The whence argument is not a proper value, or the resulting file offset would be invalid.
RFS_EBADOP	Bad operation. Seek not allowed with the flags used to open the file.
RFS_EDEVICE	Device I/O error

7.18.5 Fchmod

7.18.5.1 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_FCHMOD_RSP. Also the file descriptor used for this initiated operation is returned.

The possible values for 'result' are:

id	Definition
RFS_E0K	Ok
RFS_EBADFD	The fd argument is not a valid file descriptor.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_ENOTDIR	A component of the path prefix is not a directory.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EBADNAME	Object's name contains illegal characters.
RFS_ENOTAFILE	Object is not a file.
RFS_ENOENT	No such file or directory.
RFS_EINVALID	Bad mod option
RFS_ELOCKED	The file is locked (already opened in a conflicting mode).

7.18.6 Fstat

7.18.6.1 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_FSTAT_RSP. Also the file descriptor used for this initiated operation is returned.

The possible values for 'result' are:

id	Definition
RFS_EOK	Ok.
RFS_ENOENT	Object not found.
RFS_ENOTDIR	A component of the path prefix is not a directory.
RFS_EBADFD	Bad file descriptor.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Object's name contains illegal characters.

7.18.7 Fsync

7.18.7.1 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_FSYNC_RSP. Also the file descriptor used for this initiated operation is returned.

The possible values for 'result' are:

id	Definition
RFS_EOK	Ok.
RFS_EBADFD	Invalid file descriptor.
RFS_ENOSPACE	Out of data space.
RFS_EFSFULL	File system full, no free inodes.
RFS_EDEVICE	Device I/O error

7.18.8 Open

The T_RFS_OPEN_REQ_MSG message can be used to open a file. This message is similar to the rfs_open_nb() function. One exception is the parameter: pair_id. This parameter is returned by the non-blocking function and is handed over to operation, which handles the request via this request message. The RFS responds with a T_RFS_READY_RSP_MSG message, with command index: T_RFS_OPEN_RSP. Also the pair_id, provided via the request message, is returned via the response message and can be used for pairing the handled request with a response.

7.18.8.1 T_RFS_OPEN_REQ_MSG

```
typedef struct {
T_RV_HDR          hdr;
const char        pathname;
T_RFS_FLAGS       flags;
T_RFS_MODE        mode;
T_RFS_RET         pair_id,
T_RV_RETURN       return_path;
} T_RFS_OPEN_REQ_MSG
```

7.18.8.2 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_OPEN_RSP.

The possible values for 'result' are:

id	Definition
(Positive value)	File descriptor of file opened.
RFS_EEXISTS	An object of the same name already exists.
RFS_EACCES	- Search permission is denied for a component of the path prefix. - or the required permissions (for reading and/or writing) are denied for the given flags. - or RFS_O_CREAT is specified, the file does not exist, and the directory in which it is to be created does not permit writing.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Object's name contains illegal characters.
RFS_ENUMFD	Max number of used file descriptors reach

RFS_ENOENT	No such file or directory.
RFS_ENOTDIR	A component of the path prefix is not a directory.
RFS_EINVALID	Bad open flag options.
RFS_ELOCKED	The file is locked (already opened for writing, in a conflicting mode).
RFS_EMOUNT	Invalid mount point

7.18.9 Chmod

The T_RFS_CHMODE_REQ_MSG message can be used to open a file. This message is similar to the rfs_chmod_nb() function. One exception is the parameter: pair_id. This parameter is returned by the non-blocking function and is handed over to operation, which handles the request via this request message. The RFS responds with a T_RFS_READY_RSP_MSG message, with command index: T_RFS_CHMOD_RSP. Also the pair_id, provided via the request message, is returned via the response message and can be used for pairing the handled request with a response.

7.18.10 T_RFS_CHMOD_REQ_MSG

```
typedef struct {
T_RV_HDR          hdr;
const char        *pathname;
T_RFS_MODE        mode;
T_RFS_RET         pair_id,
T_RV_RETURN       return_path;
} T_RFS_CHMOD_REQ_MSG
```

7.18.11 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_CHMOD_RSP.

The possible values for ‘result’ are:

id	Definition
RFS_E0K	Ok
RFS_EBADFD	The fd argument is not a valid file descriptor.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_ENOTDIR	A component of the path prefix is not a directory.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EBADNAME	Object's name contains illegal characters.
RFS_ENOTAFILE	Object is not a file.
RFS_ENOENT	No such file or directory.
RFS_EINVALID	Bad mod option
RFS_ELOCKED	The file is locked (already opened in a conflicting mode).
RFS_EMOUNT	Invalid mount point

7.18.12 Stat and Lstat

The T_RFS_STAT_REQ_MSG message can be used to open a file. This message is similar to the rfs_stat_nb () function. One exception is the parameter: pair_id. This parameter is returned by the non-blocking function and is handed over to operation, which handles the request via this request message. The RFS responds with a T_RFS_READY_RSP_MSG message, with command index: T_RFS_STAT_RSP. Also the pair_id, provided via the request message, is returned via the response message and can be used for pairing the handled request with a response.

The T_RFS_LSTAT_REQ_MSG message can be used to obtain information about the file, directory or device. This message is similar to the rfs_lstat_nb() function.

7.18.12.1 T_RFS_STAT_REQ_MSG

typedef struct {

```
        T_RV_HDR    hdr;
        const char      *pathname;
        T_RFS_STAT   *stat;
        T_RFS_RET    pair_id,
        T_RV_RETURN    return_path;
  } T_RFS_STAT_REQ_MSG
```

7.18.12.2 T_RFS_LSTAT_REQ_MSG

```
typedef struct {
        T_RV_HDR            hdr;
        const char          *pathname;
        T_RFS_STAT          *stat;
        T_RV_RETURN         *return_path;
} T_RFS_LSTAT_REQ_MSG
```

7.18.12.3 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_STAT_RSP.

The possible values for ‘result’ are:

id	Definition
RFS_EOK	Ok.
RFS_ENOENT	Object not found.
RFS_ENOTDIR	A component of the path prefix is not a directory.
RFS_EBADFD	Bad file descriptor.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EMOUNT	Invalid mount point
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Object's name contains illegal characters.

7.18.13 Remove

The T_RFS_REMOVE_REQ_MSG message can be used to open a file. This message is similar to the rfs_remove_nb () function. One exception is the parameter: pair_id. This parameter is returned by the non-blocking function and is handed over to operation, which handles the request via this request message. The RFS responds with a T_RFS_READY_RSP_MSG message, with command index: T_RFS_REMOVE_RSP. Also the pair_id, provided via the request message, is returned via the response message and can be used for pairing the handled request with a response.

7.18.13.1 T_RFS_REMOVE_REQ_MSG

```
typedef struct {
        T_RV_HDR            hdr;
        const char          *pathname;
        T_RFS_RET           pair_id,
        T_RV_RETURN         return_path;
} T_RFS_REMOVE_REQ_MSG
```

7.18.13.2 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_REMOVE_RSP.

The possible values for ‘result’ are:

id	Definition
RFS_EOK	Ok.
RFS_ENOENT	File was not found.
RFS_ENOTDIR	A component of the path is not a directory.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Object's name contains illegal characters.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EACCES	File could not be removed (read-only).
RFS_ELOCKED	The file is open
RFS_EDEVICE	Device I/O error
RFS_EMOUNT	Invalid mount point
RFS_ENOTEMPTY	The named directory contains files other than '.' and '..' in it.

7.18.14 Rename

The T_RFS_RENAME_REQ_MSG message can be used to open a file. This message is similar to the rfs_rename_nb () function. One exception is the parameter: pair_id. This parameter is returned by the non-blocking function and is handed over to operation, which handles the request via this request message. The RFS responds with a T_RFS_READY_RSP_MSG message, with command index: T_RFS_RENAME_RSP. Also the pair_id, provided via the request message, is returned via the response message and can be used for pairing the handled request with a response.

7.18.14.1 T_RFS_RENAME_REQ_MSG

```
typedef struct {
T_RV_HDR            hdr;
const char          *oldname;
const char          *newname;
T_RFS_RET           pair_id,
T_RV_RETURN         return_path;
} T_RFS_RENAME_REQ_MSG
```

7.18.14.2 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_RENAME_RSP.

The possible values for 'result' are:

Id	Definition
RFS_EOK	Ok.
RFS_ENOENT	Oldname object does not exist.
RFS_EEXISTS	Newname object already exists.
RFS_EACCES	Object could not be modified (read-only).
RFS_ENOTDIR	A component of the path is not a directory.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Object's name contains illegal characters.
RFS_EFSFULL	Failed to allocate an inode for the changed object.
RFS_ENOTALLOWED	Renaming is not allowed (another new path contains another mountpoint)

7.18.15 Mkdir

The T_RFS_MKDIR_REQ_MSG message can be used to open a file. This message is similar to the rfs_mkdir_nb () function. One exception is the parameter: pair_id. This parameter is returned by the non-blocking function and is handed over to operation, which handles the request via this request message. The RFS responds with a T_RFS_READY_RSP_MSG message, with command index: T_RFS_MKDIR_RSP. Also the pair_id, provided via the request message, is returned via the response message and can be used for pairing the handled request with a response.

7.18.15.1 T_RFS_MKDIR_REQ_MSG

```
typedef struct {
        T_RV_HDR            hdr;
        const char          *pathname;
        T_RFS_MODE          mode;
        T_RFS_RET           pair_id,
        T_RV_RETURN         return_path;
} T_RFS_MKDIR_REQ_MSG
```

7.18.15.2 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_MKDIR_RSP.

The possible values for 'result' are:

id	Definition
RFS_EOK	Ok.
RFS_EEXISTS	Directory already exists.
RFS_ENOTDIR	A component of the path is not a directory.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EBADNAME	Name of the directory contains illegal characters
RFS_ENOSPACE	Failed to allocate space for object's data.
RFS_EFSFULL	Failed to allocate an inode for the object.
RFS_EDEVICE	Device I/O error
RFS_EMOUNT	Invalid mount point

7.18.16 Rmdir

The T_RFS_RMDIR_REQ_MSG message can be used to open a file. This message is similar to the rfs_rmdir_nb () function. One exception is the parameter: pair_id. This parameter is returned by the non-blocking function and is handed over to operation, which handles the request via this request message. The RFS responds with a T_RFS_READY_RSP_MSG message, with command index: T_RFS_RMDIR_RSP. Also the pair_id, provided via the request message, is returned via the response message and can be used for pairing the handled request with a response.

7.18.16.1 T_RFS_RMDIR_REQ_MSG

```
typedef struct {
        T_RV_HDR        hdr;
        const char      *pathname;
        T_RFS_RET       pair_id,
        T_RV_RETURN     return_path;
} T_RFS_RMDIR_REQ_MSG
```

7.18.16.2 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_RMDIR_RSP.

The possible values for 'result' are:

id	Definition
RFS_EOK	Ok.
RFS_ENOENT	The named directory does not exist
RFS_ENOTDIR	A component of the path is not a directory.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_ENOTEMPTY	The named directory contains files other than '.' and '..' in it.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_EACCES	No write permission to delete the directory entry
RFS_EBUSY	The directory to be removed is the mount point for a mounted file system or the current directory.
RFS_EBADNAME	Name of the directory contains illegal characters
RFS_EMOUNT	Invalid mount point

7.18.17 Opendir

The T_RFS_OPENDIR_REQ_MSG message can be used to open a file. This message is similar to the rfs_opendir_nb () function. One exception is the parameter: pair_id. This parameter is returned by the non-blocking function and is handed over to operation, which handles the request via this request message. The RFS responds with a T_RFS_READY_RSP_MSG message, with command index: T_RFS_OPENDIR_RSP. Also the pair_id, provided via the request message, is returned via the response message and can be used for pairing the handled request with a response.

7.18.17.1 T_RFS_OPENDIR_REQ_MSG

```
typedef struct {
        T_RV_HDR            hdr;
        const char          *pathname;
        T_RFS_DIR           *dirp,
        T_RFS_RET           pair_id,
        T_RV_RETURN                 return_path;
} T_RFS_OPENDIR_REQ_MSG
```

7.18.17.2 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_OPENDIR_RSP.

The possible values for ‘result’ are:

id	Definition
(Positive value)	Number of objects in the specified directory (At the time of call).
RFS_ENOTDIR	A component of the path is not a directory.
RFS_ENOENT	Directory not found.
RFS_EACCES	Search permission is denied for a component of the path prefix.
RFS_ENAMETOOLONG	Object's name is too long.
RFS_EBADNAME	Name of the directory contains illegal characters
RFS_ENOSPACE	Out of data space.
RFS_EMOUNT	Invalid mount point

7.18.18 Readdir

The T_RFS_READDIR_REQ_MSG message can be used to open a file. This message is similar to the rfs_readdir_nb () function. One exception is the parameter: pair_id. This parameter is returned by the non-blocking function and is handed over to operation, which handles the request via this request message. The RFS responds with a T_RFS_READY_RSP_MSG message, with command index: T_RFS_READDIR_RSP. Also the pair_id, provided via the request message, is returned via the response message and can be used for pairing the handled request with a response.

7.18.18.1 T_RFS_READDIR_REQ_MSG

```
typedef struct {
T_RV_HDR            hdr;
T_RFS_DIR           *dirp;
char                *buf
T_RFS_SIZE          size
T_RFS_RET           pair_id,
T_RV_RETURN         return_path;
} T_RFS_READDIR_REQ_MSG
```

7.18.18.2 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_READDIR_RSP.

The possible values for 'result' are:

id	Definition
(Positive value)	Number of bytes actually read.
RFS_EBADDIR	Invalid directory descriptor
RFS_ENOSPACE	Out of data space.

7.18.19 Preformat

The T_RFS_PREFORMAT_REQ_MSG message can be used to open a file. This message is similar to the rfs_preformat_nb () function. One exception is the parameter: pair_id. This parameter is returned by the non-blocking function and is handed over to operation, which handles the request via this request message. The RFS responds with a T_RFS_READY_RSP_MSG message, with command index: T_RFS_PREFORMAT_RSP. Also the pair_id, provided via the request message, is returned via the response message and can be used for pairing the handled request with a response.

7.18.19.1 T_RFS_PREFORMAT_REQ_MSG

```
typedef struct {
        T_RV_HDR            hdr;
        UINT16              magic;
        T_RFS_RET           pair_id,
        T_RV_RETURN         return_path;
} T_RFS_PREFORMAT_REQ_MSG
```

7.18.19.2 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_PREFORMAT_RSP.

The possible values for 'result' are:

id	Definition
RFS_EOK	Ok.
RFS_EMAGIC	Magic number is incorrect.
RFS_EINVALID	An erase operation is currently in progress. Retry the operation again later.
RFS_ENODEVICE	The flash device is unknown (not supported).
RFS_EMEMORY	Message allocation failed.
RFS_EMSGSEND	Message sending failed.
RFS_EDEVICE	Device I/O error
RFS_ENOTALLOWED	Pre-formatting is not allowed (pathname is no mountpoint)

7.18.20 Format

The T_RFS_FORMAT_REQ_MSG message can be used to open a file. This message is similar to the rfs_format_nb () function. One exception is the parameter: pair_id. This parameter is returned by the non-blocking function and is handed over to operation, which handles the request via this request message. The RFS responds with a T_RFS_READY_RSP_MSG message, with command index: T_RFS_FORMAT_RSP. Also the pair_id, provided via the request message, is returned via the response message and can be used for pairing the handled request with a response.

7.18.20.1 T_RFS_FORMAT_REQ_MSG

```
typedef struct {
        T_RV_HDR            hdr;
        const char          *name;
        UINT16              magic;
        T_RFS_RET           pair_id,
        T_RV_RETURN         return_path;
} T_RFS_FORMAT_REQ_MSG
```

7.18.20.2 Response message

The RFS responds with T_RFS_READY_RSP_MSG and contains command index: RFS_FORMAT_RSP.

The possible values for 'result' are:

id	Definition
RFS_EOK	Ok.
RFS_EAGAIN	Previous RFS_preformat() has not finished yet.
RFS_EINVALID	Magic number is incorrect.
RFS_EBADNAME	Name contains illegal characters
RFS_EMEMORY	Message allocation failed.
RFS_EMSGSEND	Message sending failed.
RFS_EDEVICE	Device I/O error
RFS_ENOTALLOWED	Formatting is not allowed (pathname is no mountpoint)

7.18.21 Generic response message

The RFS defines one generic response message T_RFS_READY_RSP_MSG. The command index indicates the asynchronous operation, which is finished. The result value varies for each finished operation. To pair a response message with the requested operation, a pair value is returned. This can be either a valid file descriptor or unique pair ID is returned.

7.18.21.1 T_RFS_READY_RSP_MSG

```
typedef struct {
        T_RV_HDR            hdr;
        T_RFS_CMD_ID        command_id;
        T_RFS_PAIR_VALUE    pair_value;
        T_RFS_RET           result;
} T_RFS_READY_RSP_MSG
```

Type's definition

7.18.22 T_RFS_CMD_ID

This defines the index of the asynchronous operation (command), which is finished.

```
typedef enum{
RFS_CLOSE_RSP,
RFS_WRITE_RSP,
RFS_READ_RSP,
RFS_LSEEK_RSP,
RFS_FCHMOD_RSP,
RFS_FSTAT_RSP,
RFS_FSYNC_RSP,
RFS_OPEN_RSP,
RFS_CHMOD_RSP,
RFS_STAT_RSP,
RFS_REMOVE_RSP,
RFS_RENAME_RSP,
RFS_MKDIR_RSP,
RFS_RMDIR_RSP,
```

```
RFS_OPENDIR_RSP,
RFS_READDIR_RSP,
RFS_PREFORMAT_RSP,
RFS_FORMAT_RSP
}T_RFS_CMD_ID;
```

7.18.23 T_RFS_PAIR_VALUE

This defines the union containing a file descriptor and unique pair ID. Depending on the request operation the valid member should be used.

```
typedef union {
        T_RFS_FD    fd;
        T_RFS_RET    pair_id
} T_RFS_PAIR_VALUE;
```

7.18.24 T_RFS_FLAGS

This defines the flag type and values. The open mode is established as a combination of the bits defined below. Of the following first three values, only one can be set. The other flags can be OR'ed to one of these first three values. When a file is opened for write-only it can not be read and when a file is opened for read only, it can not be written.

```
typedef UINT16 T_RFS_FLAGS;

#define RFS_O_RDONLY      0x00
#define RFS_O_WRONLY      0x01
#define RFS_O_RDWR 0x02
#define RFS_O_CREAT       0x04
#define RFS_O_APPEND      0x08
#define RFS_O_TRUNC       0x10
```

Additional information about the flag values:

id	Definition
RFS_O_RDONLY	File is opened as read only.
RFS_O_WRONLY	File is opened as write only
RFS_O_RDWR	File is opened for reading and writing.
RFS_O_APPEND	If set, the file offset shall be set to the end of the file prior to each write.
RFS_O_CREAT	If the file exists, this flag has no effec. Otherwise, the file shall be created
RFS_O_TRUNC	In case the file is opened in writing mode, the file size will be set to zero (file pointer point set to begin of the file). When opened in read only mode, this flag will be ignored.

7.18.25 T_RFS_MODE

Defines the mode attribute and values. The mode is an attribute of a file indicating the permission bits. The mode attribute is formed by OR'ing the values below.

```
typedef UINT16 T_RFS_MODE;

#define S_IXUSR      0x0100 // Execute permission for the user
#define S_IWUSR      0x0200 // Write permission for the user
#define S_IRUSR      0x0400 // Read permission for the user
#define S_IRWXU      0x0700 // Read Write permission mask (default) for user

#define S_IXGRP      0x0010 // Execute permission for group
#define S_IWGRP      0x0020 // Write permission for group
#define S_IRGRP      0x0040 // Read permission for group
#define S_IRWXG      0x0070 // Read Write permission mask (default) for group
```

```
#define S_IXOTH     0x0001 // Execute permission for others
#define S_IWOTH     0x0002 // Write permission for others
#define S_IROTH     0x0004 // Read permission for others
#define S_IRWXO     0x0007 // Read Write permission mask (default) for  others
```

7.18.26 T_RFS_FD

Defines the file descriptor.

```
typedef INT16 T_RFS_FD;
```

7.18.27 T_RFS_SIZE

Defines the size type.

```
typedef INT32 T_RFS_SIZE;
```

7.18.28 T_RFS_OFFSET

Defines the offset type.

```
typedef INT32 T_RFS_OFFSET;
```

7.18.29 T_RFS_STAT

Defines the overall statistics type, which contains information (meta-data) about the status of a file system or mount point or file/directory.

```
typedef union {
        T_RFS_FS_STAT           file_system;
        T_RFS_MP_STAT           mount_point;
        T_RFS_FILE_DIR_STAT     file_dir;
} T_RFS_STAT;
```

7.18.30 T_RFS_FS_STAT

Defines the structure containing information (meta-data) about the statistics of a file system

```
typedef struct {
        UINT16 oname_length;        // Maximum length object names in characters
        UINT16 pname_length;        // Maximum length path names in characters
        UINT8   max_openfiles;  // Maximum number of open files for all device
        // (static number)
        UINT8   max_opendirs;       // Maximum number of open directories for all
        // devices (static number)
        UINT8   cur_openfiles;   // Number of files currently opened for all devices
        UINT8   cur_opendirs;       // Number of directories currently opened for all
        // devices
} T_RFS_FS_STAT;
```

7.18.31 T_RFS_MP_STAT

Defines the structure containing information (meta-data) about the statistics of a mount point

```
typedef struct {
        UINT32          dev;                    // Device identification, the device serial
        // number if available.
```

```
        UINT32          read_speed;             // Read speed in Kb/s.
                                                // Measurement conditions: average speed,
        // 20 blocks, 100 Kb, clock-cycle 12 MHz
        UINT32          write_speed;            // Write speed in Kb/s
        // Measurement conditions: average speed,
        // 20 blocks, 100 Kb, clock-cycle 12 MHz
        UINT32          max_fsize;              // Maximum file size (in Kb)
        UINT32          dev_size;               // Total device size (in Kb)
        UINT32          free_space;             // Available space for storage on media's
        // partition (in Kb)
        UINT32          partition_size;   // Total partition size (in Kb)
        UINT32          used_size;              // Used partition size (in Kb)
        T_RFS_MODE      mode;                   // Object permission (ugo)
        UINT8           max_mp_openfiles;       // Maximum number of open files per
        // mount point
        UINT8           max_mp_opendirs;        // Maximum number of open directories per
        // mountpount
        char            fs_type[16];            // Type of media (string format)
        char            media_type[16];// Type of file system  (string format)
} T_RFS_MP_STAT;
```

7.18.32 T_RFS_FILE_DIR_STAT

Defines the structure containing information (meta-data) about the statistics of a file/directory.

```
typedef struct {
        UINT32          ino;            // Object inode number (unique id)
        UINT32          size;           // Object size in bytes
        time_t          mtime;          // Last modification time
        time_t          ctime;          // Last status change time
        T_RFS_MODE      mode;           // Object permission (ugo)
} T_RFS_FILE_DIR_STAT;
```

Note:
- The type time_t is a definition extracted from the C-Library and specifies a type containing time and date information.
- If *mtime* is supported, *ctime* needs also to be supported. If *mtime* is not supported, it is returned the same as *ctime.* If *ctime* is not supported they are both returned as 0.

7.18.33 T_RFS_DIR

Defines the DIR type. This represents a *directory stream*, which is an ordered sequence of all the directory entries (files) in a particular directory.

```
typedef struct {
        UINT32          opendir_ino;       // inode of directory that was opened
        UINT32          lastread_ino;  // last inode of the read directory entry
} T_RFS_DIR;
```

7.18.34 Error definitions

Below is a list of all defined RFS exceptions. All exceptions returned are of type T_RFS_RET unless otherwise stated. All exception codes, except RFS_EOK, are negative. In order to provide source compatibility with future version of RFS, the application programmer should only check for exceptions by testing if the return code is less than zero. A zero or positive return code should be treated as a success indication unless otherwise noted.

Error Definition	Description

RFS_EOK	Ok
RFS_ENODEVICE	flash device unknown
RFS_EAGAIN	not ready, try again later
RFS_ENOSPACE	out of file space (no free data space)
RFS_EFSFULL	file system full (no free inodes)
RFS_EBADNAME	bad filename
RFS_ENOENT	object not found
RFS_EEXISTS	object exists
RFS_EACCES	object access permission violation
RFS_ENAMETOOLONG	filename too long
RFS_EINVALID	invalid argument
RFS_ENOTEMPTY	directory not empty
RFS_ENOTDIR	object is not a directory
RFS_EFBIG	file too big
RFS_ENOTAFILE	object is not a file
RFS_ENUMFD	Max number of used file descriptors reach
RFS_EBADFD	Bad file descriptor
RFS_EBADDIR	Bad directory descriptor
RFS_EBADOP	Bad operation
RFS_ELOCKED	The file is locked (in use)
RFS_EMOUNT	Invalid mount point
RFS_EDEVICE	Device I/O error
RFS_EBUSY	Resource busy
RFS_ENOTADIR	object is not a directory
RFS_EMAGIC	magic number is incorrect.
RFS_EMEMORY	message allocation failed.
RFS_EMSGSEND	message sending failed.
RFS_ENOTALLOWED	Function is not allowed

7.19 Configuration Items

7.19.1 Constants

7.19.1.1 RFS_MAX_NR_OPEN_FILES

The following constant defines maximum number of currently opened files within RFS

```
RFS_MAX_NR_OPEN_FILES
```

The default number will be 10. This number is configurable to enable the customer to change the maximum number of currently opened files within RFS.

7.19.1.2 RFS_MAX_NR_OPEN_DIRS

The following constant defines maximum number of currently opened directories within the RFS.

```
RFS_MAX_NR_OPEN_DIRS
```

The default number will be 10. This number is configurable to enable the customer to change the maximum number of currently opened directories within RFS.

Chapter 8 LFS

8.1 Introduction

The purpose of the linear FFS is to be able to provide direct access to the data stored in the flash by the means of a direct read through a pointer. This is not possible in the existing FFS due to the segmentation of files through the iNode/chunk structure.

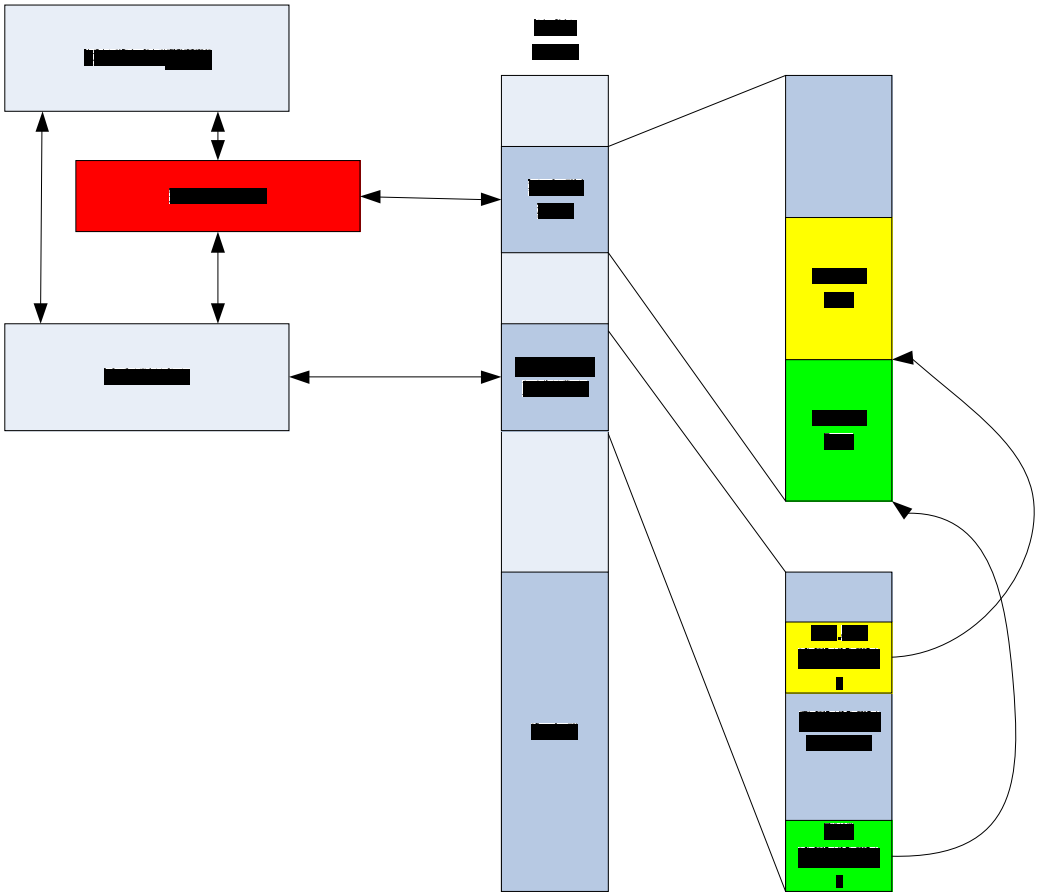


Figure 9 API of LFS

The Linear file system will allow creation of files that could fit on one physical block so that the file could be read using a pointer. As such the maximum size of a linear file system will be restricted to a little less than 64 KB. The file can be written to only once, that is at the time of creation. Once the file is closed after being created and written to, it cannot be opened for writing again. The interfaces provided by LFS are described in the next section.

8.2 Interface description

LFS will provide 5 APIs for opening/creating, reading, writing, closing and removing a linear file. A linear file will reside on a single block and hence it's maximum size will be a little less than 64 KB. Reading an LFS file will be fatster because no intermediate buffering is required. The flash is read directly using a pointer. The description of the APIs is given below. The actual APIs called by the user are wrapper APIs.

8.3 Service functions definition

8.3.1 lfs_open

```
fd_t            lfs_open (          const char *pathname,
                                    ffs_options_t option,
                                    unsigned int *size_of_file    )
```

Description

lfs_open ( ) can be used to create a new LFS file as well as open an existing one for reading. If the option FFS_O_CREATE is specified, then a new linear file will be created and its size will be equal to the passed value in the size_of_file pointer. The file will not be created if it already exists and an error will be returned. If the size of the file is >= 64KB, then also an error will be returned. The reserved field of the inode structure will be assigned a value of 0xee to denote that it is an LFS file. This field is used by ffs_open ( ) to check if the user wants to open an LFS file, and if that is the case then return an error value.

The LFS file can be written to only once, that is, at the time of creation and not after that. The user needs to call lfs_write ( ) just after lfs_open ( ) to write to the file. The option FFS_O_CREATE is stored in the returned file descriptor so that it could be checked in lfs_write ( ), and lfs_write ( ) will proceed only if the option is FFS_O_CREATE, otherwise it will return an error.

If the option is FFS_O_RDONLY, then if the file exists, a file descriptor is returned that should then be passed to lfs_read ( ) to get the pointer for reading the file. Additionally, the size of the file is also passed in the size_of_file pointer so that the application may know how much to read.

Parameters

- **pathname**

Null terminated string containing the unique name of the file to open or create.

- **option**

Specifies the attribute used to open the file.

- **size_of_file**

size_of_file will specify size of file and will be used when the FFS_O_CREAT option is specified in option.

Immediate Return

- **fd_t**

The possible values are:

id	Definition
(Positive value)	File descriptor of file opened.
EFFS_INVALID	Minimum one of the flags read_only or create must be specified.
EFFS_ACCESS	It is not a LFS file.
EFFS_NOTAFILE	It's not a valid file name.
EFFS_EXISTS	File exists the option should be FFS_O_RDONLY.
EFFS_NOTFOUND	Try to open for non existing file for operation and option filed should not be FFS_O_CREATE.
EFFS_NOSPACE	The available memory within the file system is insufficient to allocate for new file.
EFFS_NUMFD	Too many open files in system.

8.3.2 lfs_write

```
int                     lfs_write( fd_t fdi,
                  void *src,
                  int amount )
```

Description

If the file descriptor is valid and the open option used was FFS_O_CREATE, then the amount numbers of bytes are written to the flash. lfs_write () can be called many times in succession. Irrespective of the number of bytes passed by the user, the number of bytes actually written is not greater than the size specified while creating the file.

Parameters

- **fdi**

File descriptor of opened file.

- **src**

Source data buffer address.

- **amount**

Data buffer size in number of bytes.

Immediate Return

- **int**

The possible values are:

id	Definition
(Positive value)	Number of bytes written to a file.
EFFS_INVALID	Invalid parameter value.
EFFS_BADFD	Invalid file descriptor value.

8.3.3 lfs_read

```
req_id_t    lfs_read( fd_t fdi,
                      char **src )
```

Description

If the file descriptor is valid and the open option used was FFS_O_RDONLY, then a pointer to the flash address from where to start reading is passed back in the src parameter.

EFFS_OK on success else an error value. The pointer to be used for reading is passed in src.

Parameters

- **fdi**

File descriptor of opened file.

- **src**

Pointer to the flash address from where to start reading.

Immediate Return

- **req_id_t**

The possible values are:

id	Definition
EFFS_OK	The API function was successfully executed.
EFFS_BADFD	Invalid file descriptor value.
EFFS_AGAIN	Already an erase or write in progress.
EFFS_INVALID	Invalid option flag, it should be FFS_O_RDONLY

8.3.4 lfs_close

```
effs_t      lfs_close( fd_t fdi )
```

Description

If the file descriptor is valid then the file descriptor is released and EFFS_OK is returned else EFFS_BADFD is returned.

Parameters

- **fdi**

File descriptor of opened file.

Immediate Return

- **effs_t**

The possible values are:

id	Definition
EFFS_OK	The API function was successfully executed.

EFFS_BADFD	Invalid file descriptor value.

8.3.5 lfs_remove

```
effs_t      lfs_remove (      const char *pathname)
```

Description

If the file exists and if it is not already in use then the file is removed from the file system.

Parameters

- **pathname**

Null terminated string containing the unique name of the file to remove.

Immediate Return

- **effs_t**

The possible values are:

id	Definition
EFFS_OK	The API function was successfully executed.
EFFS_LOCKED	File locked, not able to remove.

Chapter 9 FFS

9.1 Introduction

This document is the programmer's manual for FFS. FFS is a flash file system with an API and implementation loosely modeled after and inspired by the POSIX file I/O interface. Objects in FFS are hierarchically organized in directories and sub-directories. FFS is crash resilient, meaning that it is able to cleanup and recover after a power failure.

This FFS interface intends to provide an easy access to FFS for applications; it gathers the applications requirements for a common File System.

9.2 Interface description

This chapter describes the API of FFS. For each function input parameters and output results are defined and described. All applications using FFS should include the `ffs.h` include file where all exception codes, data structures and functions are defined.

All functions with suffix '_nb' are non-blocking. All other functions are blocking unless otherwise noted.

All object path in the file system are absolute path, there is no relative path and no working directory concept. The path does not have to specify a volume name (not "C:/msg/inbox", but "/msg/inbox").

9.3 FFS Callback Related Types

A pointer to the following structure is supplied to all callback-enabled FFS functions (suffix `_nb`). The name is derived from `CoNFirmationPATH` and denotes the path through which, the caller wants confirmation of the operation.

In case the caller of an FFS modify-function specified the mail or function callback mechanism, a pointer to the following structure is returned. The name is derived from `CoNFirmation`.

```
typedef struct {
        T_RV_HDR      header;
        int           error;
        int           request_id;
        char          *path;
} T_FFS_FILE_CNF;
```

Except if it is ffs_write_nb(), ffs_seek_nb(), ffs_truncate_nb(), ffs_fdatasync() or ffs_close_nb() where the below structure will be used. Both functions don't have the path so instead is the file descriptor (fdi) is returned.

```
typedef struct {
        T_RV_HDR      header;
        int           error;
        int           request_id;
        T_FFS_FD      fdi;
} T_FFS_STREAM_CNF;
```

where error is the exception code of FFS operation, or zero in case of success.
`path` is the full pathname of the object the operation was performed on.

In order for the caller to be able to pair a request with the corresponding confirmation, a unique identifier, request_id, is used. The request ID is returned by all non-blocking functions. See also the section "**Error! Reference source not found.**".

The format of the header is explained in RIV010 chapter 2.4.

IMPORTANT: If any of the callback types is used (mail or function callback), it is up to the receiver to de-allocate the message buffer. See examples in section **Error! Reference source not found.** and **Error! Reference source not found.**.

9.4 FFS exceptions

Below is a list of all defined FFS exceptions. All exceptions returned are of type `T_FFS_RET` unless otherwise stated. All exception codes, except `EFFS_OK`, are negative. In order to provide source compatibility with future version of FFS, the application programmer should only check for exceptions by testing if the return code is less than zero. A zero or positive return code should be treated as a success indication unless otherwise noted.

All modify functions can return `EFFS_MEMORY` and `EFFS_MSGSEND` exception codes in addition to the other codes they returned to the client assigned callback functions or mailboxes.

(3)	EFFS_OK	ok
(4)	EFFS_NODEVICE	flash device unknown
(5)	EFFS_CORRUPTED	filesystem corrupted!?
(6)	EFFS_NOPREFORMAT	FFS not preformatted
(7)	EFFS_NOFORMAT	FFS not formatted
(8)	EFFS_BADFORMAT	FFS format not recognized (too old?)
(9)	EFFS_AGAIN	not ready, try again later
(10)	EFFS_NOSYS	function not implemented
(11)	EFFS_NOSPACE	out of file space (no free data space)
(12)	EFFS_FSFULL	file system full (no free inodes)
(13)	EFFS_BADNAME	bad filename
(14)	EFFS_NOTFOUND	object not found
(15)	EFFS_EXISTS	object exists
(16)	EFFS_ACCESS	file access permission violation
(17)	EFFS_NAMETOOLONG	filename too long
(18)	EFFS_INVALID	invalid argument
(19)	EFFS_DIRNOTEMPTY	directory not empty
(20)	EFFS_NOTADIR	object is not a directory
(21)	EFFS_FILETOOBIG	file too big
(22)	EFFS_NOTAFILE	object is not a file
(23)	EFFS_PATHTOODEEP	path too deep
(24)	EFFS_TOOBIG	too big (tmffs buffer overflow)
(25)	EFFS_NUMFD	Max number of used file descriptors reach
(26)	EFFS_BADFD	Bad file descriptor
(27)	EFFS_BADOP	Bad operation
(28)	EFFS_APPEND	Append option not specified
(29)	EFFS_LOCKED	The file is locked (in use)

These two exceptions can always be returned by **any** FFS modify-function.

(30)

(31)	EFFS_MEMORY	message allocation failed
(32)	EFFS_MSGSEND	message send failed

9.5 ffs_preformat

```
T_FFS_RET ffs_preformat(UINT16 magic);
```

```
T_FFS_REQ_ID ffs_preformat_nb(UINT16 magic, T_RV_RETURN *return_path);
```

Description:

Erase **all** data in FFS. This function must be called before `ffs_format()`. WARNING: The operation **cannot** be reversed or undone. Note that depending on the underlying flash hardware, the pre-format operation can take anything from a few milliseconds to several seconds. Most flash memories in a normal environment take around one second (typical) to erase each sector. The magic number must equal the hexadecimal constant `0xDEAD`. If the magic number given by `magic` is incorrect, `EFFS_MAGIC` is returned.

Parameters:

- ***magic*** Magic value to access the function. Must be 0xDEAD.

Return value:

id	Definition
EFFS_OK	Ok.
EFFS_MAGIC	Magic number is incorrect.
EFFS_INVALID	An erase operation is currently in progress. Retry the operation again later.
(33) EFFS_NODEVICE	(34) The flash device is unknown (not supported).
(35) EFFS_MEMORY	(36) Message allocation failed.
(37) EFFS_MSGSEND	(38) Message sending failed.

9.6 ffs_format

```
T_FFS_RET ffs_format(const char *name, UINT16 magic);
T_FFS_ REQ_ID ffs_format_nb(const char *name, UINT16 magic, T_RV_RETURN *return_path);
```

Description:

Format FFS. Miscellaneous initial meta data are written to FFS and the root directory is created with an optional volume identifier or name given by the argument `name`. The name is completely ignored and has absolutely no meaning to FFS. It can not be read or retrieved later on. If `name` is the null pointer, a default name is used. Otherwise `name` is a null terminated string starting with a slash ('/') and optionally followed by some arbitrary descriptive name for the FFS volume. For an empty FFS system, this function **must** be called after `ffs_preformat()` and before any other operation on FFS. In order to avoid spurious calls of this dangerous, unrecoverable function, the `magic` number must have the hexadecimal value of `0x2BAD` to format the flash.

Parameters:

- ***name*** Null terminated string starting with a slash('/')containing the name of the FFS volume.
- ***magic*** Magic value to access the function. Must be 0x2BAD.

Return value:

(39) **id**	(40) **Definition**
(41) EFFS_OK	(42) Ok.

(43) EFFS_AGAIN	(44) Previous ffs_preformat() has not finished yet.
(45) EFFS_INVALID	(46) Magic number is incorrect.
EFFS_BADNAME	Name does not start with a slash ('/') or contains illegal characters
(47) EFFS_MEMORY	(48) Message allocation failed.
(49) EFFS_MSGSEND	(50) Message sending failed.

9.7 ffs_open

```
T_FFS_FD ffs_open(const char * pathname, T_FFS_OPEN_FLAGS flags);
T_FFS_ REQ_ID ffs_open_nb(const char * pathname, T_FFS_OPEN_FLAGS flags,
                          T_RV_RETURN *return_path);
```

Description:

This function opens or creates a file. The *pathname* is the name of the file to open or create. If the operation succeeds, a (positive) file descriptor is returned. Otherwise an error is returned (negative value).

Parameters:

- ***pathname*** Null terminated string containing the unique name of the file to open or create.
- ***flags*** Specifies the mode used to open the file:
  - FFS_O_RDONLY File is opened as read only.
  - FFS_O_WRONLY File is opened as write only.
  - FFS_O_RDWR File is opened for reading and writing.
  - FFS_O_APPEND Append on each write.
  - FFS_O_CREATE File is created if it does not exist.
  - FFS_O_EXCL Generate error if FFS_O_CREATE is also specified and the file already exists.
  - FFS_O_TRUNC If file already exists and it is opened for writing, its length is truncated to zero.

Return value:

Id	Definition
(Positive value)	File descriptor of file opened.
EFFS_EXISTS	An object of the same name already exists.
EFFS_NAMETOOLONG	Object's name is too long.
EFFS_BADNAME	Object's name contains illegal characters.
EFFS_NUMFD	Max number of used file descriptors reach
EFFS_NOTFOUND	No such file or directory.
EFFS_INVALID	Bad open flag options.

EFFS_LOCKED	The file is locked (already opened in a conflicting mode)

9.8 ffs_close

```
T_FFS_RET ffs_close(T_FFS_FD fd);
T_FFS_ REQ_ID ffs_close_nb(T_FFS_FD fd, T_RV_RETURN *return_path);
```

Note that the calling task can be suspended in several seconds by a call to the blocking function!

Description:

Close an open file.

Parameters:

- ***fd*** File descriptor obtained when the file was opened.

Return value:

Id	Definition
EFFS_OK	Ok.
EFFS_BADFD	The file argument is not a valid file descriptor.
EFFS_FILETOOBIG	The file is too big.
EFFS_NOSPACE	Out of data space.

9.9 ffs_write

```
T_FFS_SIZE ffs_write(T_FFS_FD fd, void * buf, T_FFS_SIZE size);
T_FFS_REQ_ID ffs_write_nb(T_FFS_FD fd, void * buf, T_FFS_SIZE size,
                          T_RV_RETURN *return_path);
```

Description:

Write data to the previously opened file identified by *fd.* It writes *size* bytes of data from the buffer pointed to by *buf* at the current position of the file pointer in the file. When the write operation completes, the current position of the file pointer is set to the end of the newly added data.

Parameters:

- ***fd*** File descriptor obtained when the file was opened.
- ***buf*** Pointer to buffer of data to write.
- ***size*** Number of bytes to write.

Return value:

Id	Definition
(Positive value)	Number of bytes actually written.
EFFS_BADFD	The file argument is not a valid file descriptor.
EFFS_BADOP	The file is not open for writing.
EFFS_FILETOOBIG	The file is too big.
EFFS_NOSPACE	Out of data space.

9.10 ffs_read

```
T_FFS_SIZE ffs_read(T_FFS_FD fd, void * buf, T_FFS_SIZE size);
```

Description:

Read data from the previously opened file identified by *fd.* A maximum of *size* bytes is read from the file into the buffer pointed by *buf*. When the read operation completes, the file pointer is advanced to the end of the data read. On success, the number of bytes read is returned as a positive value. Otherwise, a negative exception code is returned.

Parameters:

- ***fd*** File descriptor obtained when the file was opened.
- ***buf*** Pointer to a buffer where the data will be copied. (The size of this buffer has to be at least *size* bytes)
- **s*ize*** Maximum number of bytes to read.

Return value:

id	Definition
(Positive value)	Number of bytes actually read.
EFFS_BADFD	The fd argument is not a valid file descriptor.
EFFS_BADOP	The file is not open for reading

9.11 ffs_seek

```
T_FFS_SIZE ffs_seek (T_FFS_FD fd, T_FFS_SIZE offset, T_FFS_WHENCE whence);
T_FFS_ REQ_ID ffs_seek_nb (T_FFS_FD fd, T_FFS_SIZE offset, T_FFS_WHENCE whence,
                           T_RV_RETURN *cp);
```

Description:

Set the file pointer of an open file. It is not allowed to set the file pointer beyond the end of the file. Neither may the resulting file pointer become negative.

Note that, depending on system load, the blocking version of this function can block the caller for a substantial amount of time.

Parameters:

- ***fd*** File descriptor obtained when the file was opened.
- ***offset*** Offset (in bytes) to move the file pointer.
- ***whence*** Origin from which the move of the current position will start:
  FFS_SEEK_SET = Absolute offset from start of file.
  FFS_SEEK_CUR = Relative to current.
  FFS_SEEK_END = Absolute offset from end of file.

Return value:

id	Definition
(Positive value)	New position of the file pointer
EFFS_BADFD	The files argument is not a valid file descriptor.
EFFS_INVALID	The whence argument is not a proper value, or the resulting file offset would be invalid.
EFFS_BADOP	Bad operation. Seek not allowed with the flags used to open the file

9.12 ffs_truncate, ffs_ftruncate

```
T_FFS_RET ffs_truncate(const char *path, T_FFS_OFFSET length);
T_FFS_ REQ_ID ffs_truncate_nb(const char *path, T_FFS_OFFSET length, T_RV_RETURN *cp);
T_FFS_RET ffs_ftruncate(T_FFS_FD fd, T_FFS_OFFSET length);
T_FFS_ REQ_ID ffs_ftruncate_nb(T_FFS_FD fdi, T_FFS_OFFSET length, T_RV_RETURN *cp);
```

Description:

Truncate causes the file named by path or referenced by fd to be truncated to at most length bytes in size. If the file previously was larger than length, the extra data is lost. If it was previously shorter than length, nothing will be done. The functions do not modify the file pointer for any open files.

With ftruncate, the file must be open for writing. It is not possible to truncate an open file to a size less than the current file pointer. Otherwise EFFS_INVALID is returned.

Truncate only succeeds if the file being truncated is not already opened. Otherwise EFFS_LOCKED is returned.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the file to be truncated.
- ***fd*** File descriptor obtained when the file was opened.
- ***length*** The new requested size of the file.

Return value:

id	Definition
EFFS_OK	Ok.
EFFS_BADFD	Invalid file descriptor.
EFFS_BADNAME	Object's name contains illegal characters.
EFFS_NOTFOUND	No such file or directory.
EFFS_ACCESS	The file is not writable.
EFFS_NOTAFILE	Object is not a file.
EFFS_INVALID	The fd is not open for writing.
EFFS_LOCKED	The file is already opened.

9.13 ffs_fdatasync

```
T_FFS_RET ffs_fdatasync(T_FFS_FD fd);
T_FFS_ REQ_ID ffs_fdatasync_nb(T_FFS_FD fd, T_RV_RETURN *cp);
```

Description:

Flush the write buffer of the specified file descriptor, fd.

Parameters:

- ***fd*** File descriptor obtained when the file was opened.

Return value:

id	Definition
EFFS_OK	Ok
EFFS_BADFD	Invalid file descriptor.
EFFS_NOSPACE	Out of data space.
EFFS_FSFULL	File system full, no free inodes

9.14 ffs_stat, ffs_fstat, ffs_lstat, ffs_xlstat

```
T_FFS_RET ffs_stat(const char *pathname,  T_FFS_STAT *stat)
T_FFS_RET ffs_fstat(T_FFS_FD fd,  T_FFS_STAT *stat)
T_FFS_RET ffs_lstat(const char *pathname, T_FFS_STAT *stat);
T_FFS_RET ffs_xlstat(const char *pathname, T_FFS_XSTAT *stat);
```

Description:

Read and return object meta-data of the object with the pathname given by `pathname`. If the file exists, the meta-data is written into the structure `stat`. If the object does not exist, `EFFS_NOTFOUND` is returned. The contents of the `T_FFS_STAT and T_FFS_XSTAT` structure is documented in the `ffs.h` include file. The xstat functions return more data than the corresponding stat functions.

fstat is identical to stat, except that a file descriptor, *fd* is stated in place of *pathname*.

lstat/xlstat:
These functions are similar to `ffs_stat()` except that these functions do **not** follow symbolic links thus these can be used to read meta-data of symbolic links.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the object in the File System.
- ***fd*** File descriptor obtained when the file was opened.
- ***stat*** (Output parameter) Contains information (meta-data) about the specified object.

Return value:

id	Definition
EFFS_OK	Ok.
EFFS_NOTFOUND	Object not found.
EFFS_BADFD	Bad file descriptor.

9.15 ffs_remove

```
T_FFS_RET ffs_remove(const char *pathname)
T_FFS_ REQ_ID ffs_remove_nb(const char *pathname, T_RV_RETURN *return_path)
```

Description:

Remove the object with the pathname given by `pathname`. The pathname is a null terminated string. If the object does not exist, `EFFS_NOTFOUND` is returned. If a directory is to be removed, it must be empty, otherwise `EFFS_DIRNOTEMPTY` is returned. It is not possible to rename a file that is open.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the object.

Return value:

Id	Definition
EFFS_OK	Ok.
EFFS_NOTFOUND	Object was not found.
EFFS_ACCESS	File could not be removed (read-only).
EFFS_DIRNOTEMPTY	Directory is not empty
(51) EFFS_MEMORY	(52) Message allocation failed.
EFFS_MSGSEND	Message sending failed.
EFFS_LOCKED	The file is open

9.16 ffs_mkdir

```
T_FFS_RET ffs_mkdir(const char *pathname)
T_FFS_ REQ_ID ffs_mkdir_nb(const char *pathname, T_RV_RETURN *return_path)
```

Description:

Create a directory with the pathname given by `pathname`. The pathname is a null terminated string. All components of the path must be already existing directories. This means that it is not possible to `ffs_mkdir("/gsm/rf/tx")` if the directories `/gsm/` and `/gsm/rf` are not already created.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the directory to create.

Return value:

Id	Definition

EFFS_OK	Ok.
EFFS_EXISTS	Directory already exists.
EFFS_NAMETOOLONG	Object's name is too long.
EFFS_BADNAME	Name of the directory contains illegal characters
EFFS_NOSPACE	Failed to allocate space for object's data.
EFFS_FSFULL	Failed to allocate an inode for the object
(53) EFFS_MEMORY	(54) Message allocation failed.
(55) EFFS_MSGSEND	(56) Message sending failed.

9.17 ffs_opendir

```
T_FFS_SIZE ffs_opendir(const char *pathname, T_FFS_DIR *dir);
```

Description:

Open the directory with the pathname given by `pathname` for reading. The pathname is a null terminated string. If the directory exists the directory structure `dir` is updated and EFFS_OK is returned. The directory structure `dir` is used internally by FFS and has no meaning to the user. If the directory does not exist, `EFFS_NOTFOUND` is returned. On success, the function returns the number of objects in the directory. Otherwise, in case an exception occurred, the exception code is returned.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the directory we want to open.
- ***dir*** Internal FFS structure which has no meaning for the user and has to be used to call ffs_readdir(). The user is responsible to allocate this structure.

Return value:

Id	Definition
(Positive value)	Number of objects in the specified directory (At the time of call).
EFFS_NOTADIR	Not a directory.
EFFS_NOTFOUND	Directory not found.

9.18 ffs_readdir

```
T_FFS_SIZE ffs_readdir(T_FFS_DIR *dir, char *buf, T_FFS_SIZE size);
```

Description:

Read an entry from a directory previously opened by `ffs_opendir()`. The name of next entry in the directory is copied into the buffer pointed to by `buf`. The size of the buffer is given by `size`. A positive return value denotes that the buffer pointed to by `buf` contains the null-terminated name of the entry found. A zero is returned if there were no more entries in the directory and the buffer pointed to by

`buf` is left untouched. The function returns a negative value if an exception is encountered. In order to read all entries in a directory, `ffs_readdir()` should be called until it returns zero.

Parameters:

- ***dir*** Pointer to a T_FFS_DIR structure obtained in a previous call to ffs_opendir().
- ***buf*** (Output parameter) Pointer to a buffer which will contain the name of the directory entry.
- ***size*** Size in bytes of the buffer pointed by *buf.*

Return value:

Id	Definition
`(Positive value)`	Number of bytes actually read.
`EFFS_NOTADIR`	Not a directory.
`EFFS_NOTFOUND`	Directory not found

.

9.19 ffs_symlink

```
T_FFS_RET ffs_symlink(const char *actualpath, const char *pathname)
T_FFS_ REQ_ID ffs_symlink_nb(const char *actualpath, const char *pathname,
                             T_RV_RETURN *return_path)
```

Description:

Create a symbolic link with the pathname given by `pathname`. The pathname is a null terminated string. The pathname of the actual file is given by `actualpath`.

Parameters:

- ***actualpath*** Null terminated string containing the unique name of the symbolic link we want to create.
- ***pathname*** Null terminated string containing the name of the actual file.

Return value:

Id	Definition
`EFFS_OK`	Ok.
`EFFS_EXISTS`	Another object of the same name already exists.
`EFFS_NAMETOOLONG`	Object's name is too long.
`EFFS_BADNAME`	Name of the directory contains illegal characters
`EFFS_NOSPACE`	Failed to allocate space for object's data.
`EFFS_FSFULL`	Failed to allocate an inode for the object.
(57) `EFFS_MEMORY`	(58) `Message allocation failed.`
(59) `EFFS_MSGSEND`	(60) `Message sending failed`

9.20 ffs_readlink

```
T_FFS_SIZE ffs_readlink(const char *pathname, char *buf,  T_FFS_SIZE size)
```

Description:

Read the contents of the symbolic link with the pathname given by `pathname`. The pathname is a null terminated string. If the link exists, the contents are copied into the buffer pointed to by `buf`. The size of the buffer is given by `size`. If the link does not exist, `EFFS_NOTFOUND` is returned. On success, the function returns the number of bytes actually read. Otherwise, in case an exception occurred, the exception code is returned.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the symbolic link we want to read the content.
- ***buf*** (Output parameter) Pointer to a buffer which will contain the contents of the symbolic link.
- ***size*** Size in bytes of the buffer pointed by *buf*.

Return value:

Id	Definition
`(Positive value)`	Number of bytes actually read.
`EFFS_NOTFOUND`	The symbolic link object does not exist.
`EFFS_FILETOOBIG`	Buffer is too small to contain link content.
`EFFS_NOTAFILE`	Object is not a file

9.21 ffs_rename

```
T_FFS_RET ffs_rename(const char *oldname, const char *newname);
T_FFS_ REQ_ID ffs_rename_nb(const char *oldname, const char *newname,
                            T_RV_RETURN *return_path);
```

Description:

Rename files, directories and symbolic links. The names are the full path to the object. It is possible to move the object to a different path simple by specifying a new path in the *newname* string.
The *oldname* object must exist and the *newname* must not existe or else an error will be returned.

Parameters:

- ***oldname*** Null terminated string containing the unique name including the path of the existing object in the File System.
- ***newname*** Null terminated string containing the unique name including the path which *oldname is* desired to change name or location to.

Return value:

id	Definition
(61) EFFS_OK	(62) Ok.
EFFS_NOTFOUND	*oldname* object does not exist.
EFFS_EFFS_EXISTS	*newname* object already exists.
EFFS_ACCESS	Object could not be modified (read-only).
EFFS_NAMETOOLONG	Object's name is too long.
EFFS_BADNAME	Object's name contains illegal characters.
EFFS_FSFULL	Failed to allocate an inode for the changed object.
(63) EFFS_MEMORY	(64) Message allocation failed.
(65) EFFS_MSGSEND	(66) Message sending failed

9.22 ffs_file_write

```
T_FFS_RET ffs_file_write(const char *pathname, void *buf, T_FFS_SIZE size,
                         T_FFS_OPEN_FLAGS flags);
T_FFS_ REQ_ID ffs_file_write_nb(const char *pathname, void *buf, T_FFS_SIZE size,
                                T_FFS_OPEN_FLAGS flags, T_RV_RETURN *return_path);
```

Description:

Write the file data of the file given by the pathname `pathname`. The pathname is a null terminated string. The data are written from the buffer described by the pointer `buf` that points to the start of the data. The size of the buffer is given by `size`.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the file to create.
- ***bufr*** Pointer to buffer of data to write.
- ***size*** Number of bytes to write.
- ***flags*** Specifies the mode used to write the file:

`FFS_O_CREATE` = File is created if it does not exist.
`FFS_O_EXCL` = Generate an error if FFS_O_CREATE is also specified and the file already exists.
`FFS_O_TRUNC` = If file already exists, replace it with the new data.

Return value:

Id	Definition
EFFS_OK	Ok.
EFFS_ACCESS	File cannot be modified (read-only).
EFFS_EFFS_EXISTS	Object exists.
EFFS_NOTAFILE	Object is not a file.
EFFS_NAMETOOLONG	Object's name is too long.
EFFS_BADNAME	Object's name contains illegal characters.
EFFS_FILETOOBIG	File data size is too big.
EFFS_NOSPACE	Failed to allocate space for object's data.
EFFS_FSFULL	Failed to allocate an inode for the object.
(67) EFFS_MEMORY	(68) Message allocation failed.
(69) EFFS_MSGSEND	(70) Message sending failed

9.23 ffs_file_read, ffs_fread

```
T_FFS_SIZE ffs_file_read(const char *pathname, void *buf, T_FFS_SIZE size);
T_FFS_SIZE ffs_fread(const char *pathname, void *buf, T_FFS_SIZE size);
```

Note important: ffs_fread() is deprecated and should not be used. Use ffs_file_read() instead.

Description:

Read the entire file with the pathname given by `pathname`. The pathname is a null terminated string. If the file does not exist, `EFFS_NOTFOUND` is returned. The file data are read from the file and copied to the buffer pointed to by `buf`. The size of the buffer is given by `size`. On success, the function returns the number of bytes actually read. Otherwise, in case an exception occurred, the exception code is returned.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the file we want to read.
- ***buf*** Pointer to a buffer where the data will be copied to. (The size of this buffer has to be at least *size* bytes)
- ***size*** Size of buffer.

Return value:

Id	Definition
`(Positive value)`	Number of bytes actually read.
`EFFS_NOTFOUND`	File not found.
`EFFS_FILETOOBIG`	File data is larger than the buffer size given by `size`.
`EFFS_NOTAFILE`	The pathname refers not to a file

9.24 ffs_fcreate

```
T_FFS_RET ffs_fcreate(const char *pathname, void *buf, T_FFS_SIZE size);
T_FFS_ REQ_ID ffs_fcreate_nb(const char *pathname, void *buf, T_FFS_SIZE size,
                             T_RV_RETURN *return_path);
```

Note: `ffs_fcreate()` is deprecated and should not be used. Use `ffs_file_write(..., FFS_O_CREATE | FFS_O_EXCL)` instead.

Description:

Create a file with a pathname given by `pathname`, which is a null terminated string. If the file already exists, `EFFS_EXISTS` is returned. The file data are written from the buffer described by the pointer `buf` which points to the start of the data. The size of the buffer is given by `size`.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the file to create.
- ***buf*** Pointer to buffer of the data to write.
- ***size*** Number of bytes to write.

Return value:

Id	Definition
EFFS_OK	Ok.
EFFS_EXISTS	An object of the same name already exists.
EFFS_NAMETOOLONG	Object's name is too long.
EFFS_BADNAME	Object's name contains illegal characters.
(71) EFFS_FILETOOBIG	(72) File data size is too big.
EFFS_NOSPACE	Failed to allocate space for object's data.
EFFS_FSFULL	Failed to allocate an inode for the object.
(73) EFFS_MEMORY	(74) Message allocation failed.
(75) EFFS_MSGSEND	(76) Message sending failed

9.25 ffs_fupdate

```
T_FFS_RET ffs_fupdate(const char *pathname, void *buf, T_FFS_SIZE size);
T_FFS_ REQ_ID ffs_fupdate_nb(const char *pathname, void *buf, T_FFS_SIZE size,
                             T_RV_RETURN *return_path);
```

Note: `ffs_fupdate()` is deprecated and should not be used. Use `ffs_file_write(...,FFS_O_TRUNC)` instead.

Description:

Update the contents of the file with the pathname given by `pathname`, which is a null terminated string. If the file does not exist, `EFFS_NOTFOUND` is returned. The file data are written from the buffer described by the pointer `buf` which points to the start of the data. The size of the buffer is given by `size`.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the file to update.
- **buf** Pointer to buffer of data to write.
- ***size*** Number of bytes to write.

Return value:

Id	Definition
EFFS_OK	Ok.
EFFS_NOTFOUND	Object not found.
EFFS_ACCESS	File cannot be modified (read-only).
EFFS_NAMETOOLONG	Object's name is too long.
EFFS_BADNAME	Object's name contains illegal characters.
EFFS_FILETOOBIG	File data size is too big.

EFFS_NOSPACE	Failed to allocate space for object's data.
EFFS_FSFULL	Failed to allocate an inode for the object.
(77) EFFS_MEMORY	(78) Message allocation failed.
(79) EFFS_MSGSEND	(80) Message sending failed

9.26 ffs_fwrite

```
T_FFS_RET ffs_fwrite(const char *pathname, void *buf, T_FFS_SIZE size);
T_FFS_ REQ_ID ffs_fwrite_nb(const char *pathname, void *buf, T_FFS_SIZE size,
                            T_RV_RETURN *return_path);
```

Note: `ffs_fwrite()` is deprecated and should not be used. Use `ffs_file_write(...,FFS_O_CREATE | FFS_O_TRUNC)` instead.

Description:

Write the file data of the file given by the pathname `pathname`. The pathname is a null terminated string. If the file does not exist, it is created. If file exists, it is updated. The file data are written from the buffer described by the pointer `buf` that points to the start of the data. The size of the buffer is given by `size`.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the file to create.
- ***buf*** Pointer to buffer of data to write.
- ***size*** Number of bytes to write.

Return value:

Id	Definition
EFFS_OK	Ok.
EFFS_ACCESS	File cannot be modified (read-only).
EFFS_NAMETOOLONG	Object's name is too long.
EFFS_BADNAME	Object's name contains illegal characters.
EFFS_FILETOOBIG	File data size is too big.
EFFS_NOSPACE	Failed to allocate space for object's data.
EFFS_FSFULL	Failed to allocate an inode for the object.
(81) EFFS_MEMORY	(82) Message allocation failed.
(83) EFFS_MSGSEND	(84) Message sending failed

9.27 ffs_fcontrol

```
T_FFS_RET ffs_fcontrol(const char *pathname, INT8 type, int param);
T_FFS_ REQ_ID ffs_fcontrol_nb(const char *pathname, INT8 type, int param,
                              T_RV_RETURN *return_path);
```

Description:

Write meta-data of the object with the pathname given by `pathname`. The meta-data written is denoted by `type` and the value written is given by `param`. If an invalid type or parameter is given, `EFFS_INVALID` is returned.

Parameters:

- ***pathname*** Null terminated string containing the unique name of the object in the File System.
- ***type*** Object actions:
- OC_FLAGS = Set object flags.
- ***param*** Object flags:
- OF_READONLY = Object can not be modified or deleted.

Return value:

(85) Id	(86) Definition
(87) EFFS_OK	(88) Ok.
(89) EFFS_INVALID	(90) The specified meta-data type is unknown or param value is invalid.
EFFS_NOTFOUND	Object does not exist.
EFFS_ACCESS	Object could not be modified (read-only).
EFFS_FSFULL	Failed to allocate an inode for the changed object.
(91) EFFS_MEMORY	(92) Message allocation failed.
(93) EFFS_MSGSEND	(94) Message sending failed

(95) .

Note: ffs_fcontrol() will change name to ffs_file_control() in a future version.

9.28 ffs_query

```
T_FFS_RET ffs_query(INT8 query, void *buf);
```

Description:

Read and return the FFS parameter given by `query`. If `query` is valid, the value is read and copied to the buffer pointed to by `buf`. If an unknown query is attempted, `EFFS_INVALID` is returned. Valid queries, their size (in bytes) and description are as given below:

Parameters:

Query	Size	Description
(96) Q_BLOCKS_FREE	2	number of free blocks
(97) Q_BLOCKS_LOW	2	garbage collection parameter
(98)		
(99) Q_BYTES_FREE	4	number of free bytes in FFS
(100) Q_BYTES_USED	4	number of used bytes in FFS
(101) Q_BYTES_LOST	4	number of lost bytes in FFS
(102) Q_BYTES_MAX in FFS	4	number of max available bytes
(103)		
(104) Q_FILENAME_MAX	2	max filename length
(105)		
(106) Q_TM_BUFADDR	4	testmode buffer addr
(107) Q_TM_BUFSIZE	4	testmode ffs buffer size
(108)		
(109) Q_FFS_API_VERSION	2	FFS API Version
(110) Q_FFS_DRV_VERSION	2	FFS Driver Version
(111) Q_FFS_REVISION	2	FFS Revision
(112) Q_FFS_FORMAT_WRITE flash blocks	2	FFS version as formatted in
(113) Q_FFS_FORMAT_READ	2	FFS version as read from ffs
(114) Q_FFS_LASTERROR init)	2	FFS last exception (from
(115) Q_FFS_TM_VERSION	2	FFS testmode version
(116) Q_PATH_DEPTH_MAX depth	2	max path/directory nesting
(117)		
(118) Q_OBJECTS_FREE created	2	number of objects that can be

(119) Q_OBJECTS_MAX	2	max number of valid objects allowed
(120) Q_OBJECTS_TOTAL	2	accumulated number of valid objects in FFS
(121)		
(122) Q_INODES_MAX	2	physical total max number of inodes
(123) Q_INODES_USED	2	number of inodes used
(124) Q_INODES_LOST	2	number of inodes lost
(125) Q_INODES_HIGH	2	watermark for when inodes will be
(126) reclaimed		
(127)		
(128) Q_DEV_MANUFACTURER	2	flash manufacturer ID
(129) Q_DEV_DEVICE	2	flash device ID
(130) Q_DEV_BLOCKS	2	number of FFS blocks in device
(131) Q_DEV_ATOMSIZE	2	atomsize used by FFS for this device
(132) Q_DEV_BASE	4	FFS device base address
(133) Q_DEV_DEVICE	2	flash device ID
(134)		
(135) Q_FD_BUF_SIZE	4	size of buffer used by stream functions
(136) Q_FD_MAX	2	max number of simultaneous open files

(137)

- **buf** (Output parameter) Pointer to a buffer which will contain the contents of the query value.

Return value:

(138) Id	(139) Definition
(140) EFFS_OK	(141) Ok.
(142) EFFS_INVALID	(143) Invalid argument

9.29 ffs_is_modifiable

```
T_FFS_RET ffs_is_modifiable(const char *pathname)
```

Description:

This function is not really an FFS API function but since it is closely related to all modify functions it is mentioned here. It can be considered an intrinsic FFS API function.

This function is to be implemented by the user. The function is by default located in the file `cfgffs.c`. It is called by FFS when an application tries to modify an object which has the read-only flag set. The

user can use the null-terminated `pathname` string to determine if the object being modified should indeed be considered read-only.

The coding rules for the application programmer are:

- `ffs_is_modifiable()` must return zero in order to enforce read-only. As a result of this, the originally called FFS function will return the exception EFFS_ACCESS.
- `ffs_is_modifiable()` must return value of non-zero to allow modifications to happen. Then FFS will proceed with the operation.

Chapter 10 USB

10.1 Introduction

10.2 Interface description

This chapter is used for the ENTITY interface description. It is not required to specify the Generic interface

10.2.1 usb_fm_subscribe

```
T_RV_RET usb_fm_subscribe(UINT8 interface_id, T_RV_RETURN return_path)
```

Description

The function is called by a FM to subscribe to an interface.
Every interface is controlled by a function manager. In the "usb_interface_cfg.h" is defined which FM controls which interface. This function is used to actually subscribe the defined FM to the specified interface. Theoretically the usb driver can see which FM is subscribing by reading the "hdr". In the case that 1 FM supports more than 1 interface the FM must indicate during subscription, to which interface it will subscribe.

Parameters

- **interface_id** holds the interface number as described in "usb_interface_cfg.h"
- **return_path** return path of the function call

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	
RV_NOT_SUPPORTED	
RV_NOT_READY	
RV_MEMORY_WARNING	
RV_MEMORY_ERR	
RV_MEMORY_REMAINING	
RV_INTERNAL_ERR	
RV_INVALID_PARAMETER	

10.2.2 usb_fm_unsubscribe

```
T_RV_RET usb_fm_unsubscribe(UINT8 interface_id)
```

Description

This function must be called by a FM to release its subscription to an interface.
Every interface is controlled by a function manager. In the "usb_interface_cfg.h" is defined which FM controls which interface. This function is used to release the subscription with the defined FM to the specified interface.

Parameters

- **interface_id** holds the interface number as described in “usb_interface_cfg.h.

Immediate Return

T_RV_RET

The possible values are:

id	Definition
RV_OK	
RV_NOT_READY	
RV_INVALID_PARAMETER	

10.2.3 usb_get_status

```
T_RV_RET usb_get_status(UINT8 interface_id, T_USB_STATUS* status_p)
```

Description

This function must be called by a FM to get status info.

The FM calls this function to retrieve status information about the endpoints and the USB Hardware. The diver will respond to this function call by sending a T_RV_HDR type variable to the callback function that belongs with the requester FM.

Parameters

- **interface_id** holds the interface number as described in "usb_interface_cfg.h"
- **status_p** pointer to status information storage space created by FM, filled by usb driver.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	
RV_NOT_READY	
RV_INVALID_PARAMETER	

10.2.4 usb_set_rx_buffer

```
T_RV_RET usb_set_rx_buffer(UINT8 interface_id, UINT8 endpoint, UINT8*
buffer_p, UINT16 size)
```

Description

This function provides the usb driver with a buffer into which the received data can be placed.

The driver places all the data packets as large as the available endpoints into this buffer as one large data packet.

Parameters

- **Interface_id** holds the interface number of the specified endpoint.
- **Endpoint** endpoint number
- **buffer_p** pointer to the provided data buffer
- **size** size of the provided buffer

Immediate Return

id	Definition
RV_OK	
RV_NOT_READY	
RV_INVALID_PARAMETER	

Restriction

This function must always be called after the interface has been notified of received data.

10.2.5 usb_reclaim_rx_buffer

```
T_RV_RET usb_reclaim_rx_buffer (UINT8 interface_id, UINT8 endpoint)
```

Description

This function gives the FM back the control over the buffer.The USB expects to get a new buffer.

Parameters

- **Interface_id** holds the interface number of the specified endpoint.
- **endpoint** endpoint number

Immediate Return

- **T_RV_RET**

id	Definition
RV_OK	
RV_NOT_READY	
RV_INVALID_PARAMETER	

10.2.6 usb_set_tx_buffer

```
T_RV_RET usb_set_tx_buffer(UINT8 interface_id, UINT8 endpoint, UINT8*
buffer_p, UINT16 size,BOOL shorter_transfer)
```

Description

This function provides the usb driver with a buffer containing data to be send.The driver splits the buffer in data packets as large as the available endpoints

Parameters

- **interface_id** holds the interface number of the specified endpoint.
- **endpoint** endpoint number
- **buffer_p** pointer to the provided data buffer
- **size** size of the provided buffer

Immediate Return

- **T_RV_RET**

id	Definition
RV_OK	
RV_NOT_READY	
RV_INVALID_PARAMETER	

10.2.7 usb_get_hw_version

```
UINT8 usb_get_hw_version(void)
```

Description

This function must be called to get the USB hardware version. This function is called this to retrieve hardware version information of the USB hardware.

Parameters

void

Immediate Return

UINT8: b0-3 minor version number (4bits),
b4-7 major version number (4bits)

10.2.8 usb_get_sw_version

```
UINT32 usb_get_sw_version(void)
```

Description

This function must be called to get the USB software driver version.The function is called this to retrieve software version information of the USB driver.

Parameters

void

Immediate Return

UINT32: b0-15 build number (8bits)
b16-23 minor version number (8bits)
b24-32 major version number (8bits)

10.2.9 usb_con_int

```
void usb_con_int()
```

Description

This function is called by the ABB external interrupt handler.

This function is called to initialize the USB transceiver (TRITON - TWL3029).
In functional terms, it is responsible for bringing up the USB Connectivity.

Parameters

void

Immediate Return

void

10.2.10 usb_discon_int

```
void usb_discon_int()
```

Description

This function is called by the ABB external interrupt handler.

This function is called to disconnect the USB transceiver (TRITON - TWL3029).
In functional terms, it is responsible for bringing down the USB Connectivity.

Parameters

void

Immediate Return

void

10.3 Message definition

There are two types of messages, request messages and response messages. All the request message definitions contain return path and the response message structures contain operation id as their status information. The message definitions are located in the directory usb_message.h.

10.3.1 USB_FM_SUBSCRIBE_MSG

This message must be used by a FM to subscribe to an interface.
Every interface is controlled by a function manager. This message is used to actually subscribe the defined FM to the specified interface. Theoretically the usb driver can see which FM is subscribing by reading the "hdr". In the case that 1 FM supports more than 1 interface the FM must indicate during subscription, to which interface it will subscribe.

```
typedef struct {
        T_RV_HDR            hdr;
        UINT8               interface_id ;
        T_RV_RETURN         return_path;
} T_USB_FM_SUBSCRIBE_MSG;
```

- **hdr** hearder of the message.
- **interface_id** this variable is used by FM to indicate to which IF wants to subscribe
- **return_path** use this path to notify swe of buffer full / empty etc

10.3.2 USB_FM_UNSUBSCRIBE_MSG

This message must be used by a FM to release it subsrciption to an interface. Every interface is controlled by a function manager. This message is used to release the subsription with the defined FM to the specified interface.

```
typedef struct    {
                  T_RV_HDR                hdr;
                  UINT8                   interface_id;
                  } T_USB_FM_UNSUBSCRIBE_MSG;
```

- **hdr** Header of the message
- **interface_id** This variable is used by FM to indicate to which IF it wants to subscribe

10.3.3 USB_GET_STATUS_MSG

This message must be used by a FM to get status info.The FM send this message to retrieve status information about the endpoints and the USB Hardware. The diver will respond to this message by sending a T_USB_STATUS_READY_MSG.

```
typedef struct {
        T_RV_HDR            hdr;
        UINT8               interface_id;
        T_USB_STATUS*       status_p;
        } T_USB_GET_STATUS_MSG;
```

- **hdr** Message header.
- **interface_id** This variable holds an interface_id part of the FM that placed the get status request so that a response can be send to it.
- **status_p** pointer to status information storage space created by FM, filled by usb driver.

10.3.4 USB_SET_TX_BUFFER_MSG

This message provides the usb driver with a buffer containing data to be send.The driver splits the buffer in data packets as large as the available endpoints fifo.

```
typedef struct{
        T_RV_HDR            hdr;
        UINT8               interface_id;
        UINT8               endpoint;
        UINT8*              buffer_p;
        UINT16              size;
        BOOL                shorter_transfer;
        } T_USB_SET_TX_BUFFER_MSG;
```

- **hdr** header of the message
- **interface_id** This variable is used by FM to indicate to which IF the specified endpoint belongs.
- **endpoint** endpoint associated with the buffer
- **buffer_p** pointer to the reserved buffer
- **size** Size of buffer in bytes. The size mainly depends on the interface description.
- **shorter_transfer** shorter than expected by the host

10.3.5 USB_SET_RX_BUFFER_MSG

typedef T_USB_SET_TX_BUFFER_MSG T_USB_SET_RX_BUFFER_MSG;

10.3.6 USB_RECLAIM_RX_BUFFER_MSG

This message gives the FM back the control over the buffer.The USB expects to get a new buffer.

```
typedef struct {
        T_RV_HDR            hdr;
        UINT8               interface_id;
        UINT8                       endpoint;
        } T_USB_RECLAIM_RX_BUFFER_MSG;
```

- **hdr** Message header
- **interface_id** This variable is used by FM to indicate to which IF the specified endpoint belongs
- **endpoint** endpoint associated with the buffer

```
/*****************************************************************************************************/
```

This part describes the message definitions of messages that are sent from the usb driver to the FM. the memory claimed for those message by the usb driver will be freed by the FM.

```
/*****************************************************************************************************/
```

10.3.7 USB_FM_RESULT_MSG

This message returns whether the subscription or unsubscription was successful or not.

```
typedef struct {
        T_RV_HDR                hdr;
        T_USB_RESULT            result;
        } T_USB_FM_RESULT_MSG;
```

- **Hdr** Message header
- **result** contains the execution result of action

10.3.8 USB_BUS_CONNECTED_MSG

This message is used by the usb driver to inform the FM of a USB bus connection.The USB Driver sends this message to ALL FMs to indicate that the USB bus has been connected.

```
typedef struct {
        T_RV_HDR                hdr;
} T_USB_BUS_CONNECTED_MSG;
```

10.3.9 USB_BUS_DISCONNECTED_MSG

This message is used by the usb driver to inform the FM of a USB bus disconnection. The USB driver sends this message to ALL FMs to indicate that the USB bus has been disconnected. Thus the FM cannot transfer data anymore.

```
typedef struct {
        T_RV_HDR                hdr;
        } T_USB_BUS_DISCONNECTED_MSG;
```

10.3.10 USB_BUS_SUSPEND_MSG

This message is used by the usb driver to inform the FM that the bus bus enters the suspend state.This message is sent to ALL FMs.

```
typedef struct {
        T_RV_HDR                hdr;
} T_USB_BUS_SUSPEND_MSG;
```

10.3.11 USB_BUS_RESUME_MSG

This message is send to the interfaces that are part of the current active configuration to indicate that the USB bus is now resuming. Host to device transfers will start soon.

```
typedef struct {
        T_RV_HDR                hdr;
} T_USB_BUS_RESUME_MSG;
```

10.3.12 USB_STATUS_READY_MSG

This message is used by the usb driver to inform the FM that the requested status data is available.

```
typedef struct {
        T_RV_HDR                hdr;
} T_USB_STATUS_READY_MSG;
```

10.3.13 USB_RX_BUFFER_FULL_MSG

This message is used by the usb driver to inform the FM that the rx buffer is full.The USB driver sends this message to an FM to indicate that a the buffer of a specified endpoint of the specified interface has been filled and that it is ready to be consumed.

```
typedef struct {
        T_RV_HDR                hdr;
        UINT8                   endpoint;
        UINT8                   interface;
        UINT16                  size;
        BOOL                    end_of_packet;
        }T_USB_RX_BUFFER_FULL_MSG;
```

- **Hdr** Message Header
- **endpoint** endpoint associated with the buffer
- **interface** interface associated with the buffer
- **size** number of bytes written in buffer
- **end_of_packet** TRUE = driver has detected end of packet
  FALSE= no end of packet has been detected

10.3.14 USB_TX_BUFFER_EMPTY_MSG

This message is used by the usb driver to inform the FM that the tx buffer is empty.This message is sent by the driver to the FM that owns the endpoint to indicate that the TX buffer has been sent. The FM can now fill the buffer again.

```
typedef struct {
        T_RV_HDR            hdr;
        UINT8               endpoint;
        UINT8               interface;
} T_USB_TX_BUFFER_EMPTY_MSG;
```

- **hdr** Message Header
- **endpoint** endpoint associated with the buffer
- **interface** endpoint associated with the buffer

10.3.15 USB_TX_EP_INTERRUPT

This message is used by the usb driver to inform the FM that the tx buffer is empty. This message is sent by the driver to the FM that owns the endpoint to indicate that the TX buffer has been sent. The FM can now fill the buffer again.

```
typedef struct    {
        T_RV_HDR            hdr;
        UINT8               endpoint;
        } T_USB_TX_EP_INTERRUPT_MSG;
```

- **Hdr** Message header
- **endpoint** endpoint associated with the buffer

10.3.16 USB_TX_BUFFER_EMPTY_MSG

This message is used by the usb driver to inform the FM that the tx buffer is empty. This message is sent by the driver to the FM that owns the endpoint to indicate that the TX buffer has been sent. The FM can now fill the buffer again.

```
typedef struct {
        T_RV_HDR            hdr;
        UINT8               endpoint;
} T_USB_RX_EP_INTERRUPT_MSG;
```

- **Hdr** Message header
- **endpoint** endpoint associated with the buffer

10.3.17 USB_HOST_RESET_MSG

This message is send by the driver to signal the interfaces of the FM that the USB Host has reset the USB device. The necessary actions must be taken by the interface.

```
typedef struct {
        T_RV_HDR                hdr;
} T_USB_HOST_RESET_MSG;
```

10.4 Types definitions and constants

API type definitions and constants are located in the configuration file usb_api.h in the common directory.

10.4.1 T_RV_RET T_USB_RETURN

USB Return type and return values. Currently they are the standard RV return types, but they may be customized in the future.

```
typedef T_RV_RET T_USB_RETURN;

#define USB_OK                          RV_OK
#define USB_NOT_SUPPORTED               RV_NOT_SUPPORTED
#define USB_MEMORY_ERR                  RV_MEMORY_ERR
#define USB_INTERNAL_ERR                RV_INTERNAL_ERR
```

10.4.2 T_USB_FM_ID

USB interface information::This is type describes the class of this interface, which is just like the subclass, USB defined (compile time is known what class the interface belongs to) what the class does and how it works is defined by USB.

```
typedef struct {
        UINT8   interface_id;           /*interface class id (usb defined)*/
        UINT8   subclass_id;            /*interface subclass id*/
        }T_USB_FM_ID;
```

10.4.3 T_USB_EP_STAT

This is type describes the information that is returned on a get status request for a specific endpoint.

```
typedef enum {
        enabled=0,              /* endpoint is enabled */
        stalled,                /* endpoint is stalled */
        unassociated            /* endpoint does not belong to this interface */
        }T_USB_EP_STAT;
```

10.4.4 USB_STATUS

name USB Status typ::This is type describes the information that is returned on a get status request.

```
typedef struct {
        UINT8   active_config;
        T_USB_EP_STAT* ep_status_p;
```

```
    UINT8  nr_of_ep;
    BOOL   driver_ready;
    BOOL   usb_connected;
    }T_USB_STATUS;
```

- **active_config** number of the active configuration,-1 if no configuration is active yet.
- **ep_status_p** static list of 30 elements with status per logical endpoint.
- **nr_of_ep** number of endpoints in list assigned to the interface.-1 when interface is not part of current active configuration.
- **driver_ready** Indicates if the USB device has been configured and is ready.
- **usb_connected** Indicates if the USB is connected to an USB bus. Can be used, when a FM is started and does not know if the bus is active.

10.4.5 T_USB_RESULT

USB Status type::This is type is used to report back the result of a sucbscribe etc.

```
typedef enum {
    succes = 0,          //action performed successfully
    fail_subscribe,      //generic failure to subscribe
    fail_unsubscribe,    //generic failure to unsubscribe
    not_subscribed,      //unsubscribe while not subscribed
    config_error,        //error in configuration
    param_error,         //parameter error
    unexpected_error     //generic failure
}T_USB_RESULT
```

Chapter 11 USBFAX

11.1 Introduction

11.2 Interface description

11.2.1 dio_init_usb

```
U16   dio_init_usb (T_DIO_DRV *drv_handle)
```

Description

The function initializes USB driver.

Parameters

- **drv_handle** unique handle for DIO drivers

Immediate Return

- **U16**

The possible values are:

id	Definition
DRV_OK	When initialization successful
DRV_INITIALIZED	When driver is already initialized
DRV_INITFAILURE	On failed initialization

11.2.2 dio_export_usb

```
void dio_export_usb(T_DIO_FUNC** dio_func)
```

Description

USB driver function set export.

Parameters

- **dio_func** pointer to the list of functions exported by the driver

Immediate Return

none

11.2.3 usbfax_getdio_sw_version

```
U32 usbfax_getdio_sw_version (void)
```

Description

This function gives software version of usbfax.

Parameters

void

Immediate Return

- **U32**

Value of software version.

Chapter 12 USBMS

12.1 Introduction

12.2 Interface description

12.2.1 usbms_send_sample

```
T_USBMS_RETURN usbms_send_sample()
```

Description

Parameters

void

Immediate Return

- **T_USBMS_RETURN**

id	Definition
USBMS_OK	
USBMS_NOT_SUPPORTED	
USBMS_MEMORY_ERR	
USBMS_INTERNAL_ERR	
USBMS_ERROR	

12.3 Types definitions and constants

API type definitions and constants are located in the configuration file usb_api.h in the common directory.

12.3.1 T_USBMS_RETURN

USBMS Return type and return values. Currently they are the standard RV return types, but they may be customized in the future.

typedef T_RV_RET T_USBMS_RETURN;

#define USBMS_OK RV_OK
#define USBMS_NOT_SUPPORTED RV_NOT_SUPPORTED
#define USBMS_MEMORY_ERR RV_MEMORY_ERR
#define USBMS_INTERNAL_ERR RV_INTERNAL_ERR
#define USBMS_ERROR -12

Chapter 13 TIMER

13.1 Introduction

13.2 Interface description

13.2.1 TIMER_Read

```
SYS_UWORD16 TIMER_Read (unsigned short regNum)
```

Description

This function reads one of the TIMER register.

Parameters

- **regNum** number of the register to be read

Immediate Return

- **SYS_UWORD16**

Value of the timer register read.

13.2.2 TM_ResetTimer

```
void  TM_ResetTimer (SYS_UWORD16 timerNum,  SYS_UWORD16 countValue,
                     SYS_UWORD16 autoReload, SYS_UWORD16 clockScale)
```

Description

This function gives the timewr state

Parameters

- **timerNum** timer number (1 or 2) to be read
- **countValue** timer value
- **autoReload** reload yes or not
- **clockScale** scaling of the clock

Immediate Return

None

13.2.3 TM_StopTimer

```
void TM_StopTimer(int TimerNum)
```

Description

This function stops the timer.

Parameters

- **TimerNum** timer number (1 or 2) to be stopped

Immediate Return

None

13.2.4 TM_ReadTimer

```
SYS_UWORD16 TM_ReadTimer (int timerNum)
```

Description

This function returns current timer value.

Parameters

- **timerNum** timer number (1 or 2) to be read

Immediate Return

- **SYS_UWORD16**

Returns current timer value.

13.2.5 TM_StartTimer

```
void TM_StartTimer(int timerNum)
```

Description

This function asks the required timer to start.

Parameters

- **timerNum** timer number (1 or 2) to be started

Immediate Return

None

13.2.6 TM_DisableWatchdog

```
void TM_DisableWatchdog(void)
```

Description

This function disables watchdog timer.

Parameters

Void

Immediate Return

None

13.2.7 TM_EnableWatchdog

```
void TM_EnableWatchdog(void)
```

Description

This function enables the watchdog timer.

Parameters

Void

Immediate Return

None

13.2.8 TM_ResetWatchdog

```
void TM_ResetWatchdog(SYS_UWORD16 count)
```

Description

This function resets watchdog timer.

Parameters

- **count** Use a different value each time, otherwise watchdog bites

Immediate Return

None

13.2.9 TM_EnableTimer

```
TM_EnableTimer(int TimerNum)
```

Description

This function enables the timer.

Parameters

- **TimerNum** timer to enable (timer1 or timer2)

Immediate Return

None

13.2.10 TM_DisableTimer

```
void TM_DisableTimer(int TimerNum)
```

Description

This function disables the timer.

Parameters

- **TimerNum** timer number (1 or 2) to be disabled

Immediate Return

void

13.2.11 TIMER_ReadValue

```
unsigned short TIMER_ReadValue(void)
```

Description

This function reads the timer value.

Parameters

Void

Immediate Return

None

13.2.12 TIMER_WriteValue

```
unsigned short TIMER_WriteValue(SYS_UWORD16 value)
```

Description

This function reads the timer value.

Parameters

Void

Immediate Return

None

Chapter 14 UART FAX & DATA

14.1 Introduction

The purpose of this document is to describe the interface of the functions provided by the UART driver, when the UART device is used for Fax & Data.

This driver only allows managing the UART Modem serial device of the B-Sample, C-Sample, D-Sample and E-Sample Texas Instruments Development Boards. Indeed, the UART IrDA device of these boards can't be used for Fax & Data because the hardware flow control (either CTS/RTS for B-Sample or DCD/DTR for C-Sample, D-Sample and E-Sample) is not supported.
Moreover the UART Modem2 of the E-Sample platform cannot be used for Fax & Data because this device is not directly accessible through a DB9 connector.

The use of serial ports of Texas Instruments Development Boards (B-Sample, C-Sample, D-Sample or E-Sample) is directly managed by a software module called serialswitch .

A software application (GSM/GPRS Protocol Stack) deals with several serial data flows that are dynamically connected by the serialswitch module to the hardware serial devices through software UART drivers.

The figure below illustrates how the serial data flows of a GSM/GPRS Protocol Stack application could be linked to the hardware serial devices.

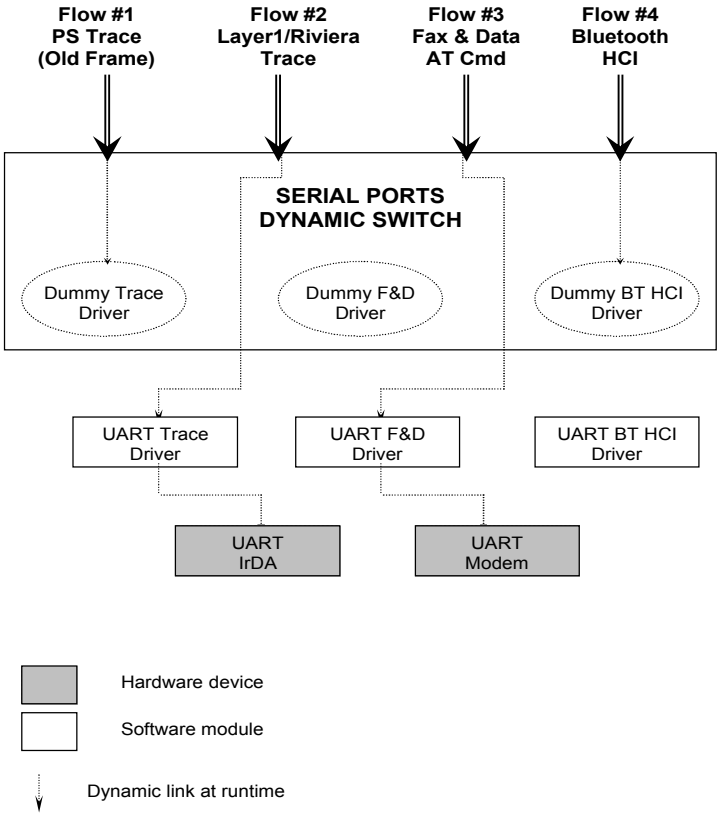


Figure 10 Data flows and HW devices

14.2 Interface description

14.2.1 UAF_Init

```
T_FDRET UAF_Init (T_fd_UartId uartNo)
```

Description

This function initializes the UART hardware used for Fax & Data, and installs the interrupt handlers. The parameters are set to the following default values:

- 19200 bps,
- 8 data bits per character,
- 1 stop bit,
- no parity,
- no flow control.

All functionalities of the UART driver are disabled, and must be enabled by calling the function UAF_Enable.

Parameters

- **uartNo** This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_INTERNAL_ERR	in case of internal problems with the hardware..

14.2.2 UAF_Enable

```
T_FDRET UAF_Enable(T_fd_uartid uartNo,SYS_BOOL enable)
```

Description

This function enables or disables the functionalities of the UART driver used for Fax & Data, according to the value of **enable**. In the deactivated state, all information about the communication parameters are stored and recalled if the driver is again enabled. When the driver is enabled the Rx and Tx buffers are cleared.

Parameters

- **uartNo** This parameter indicates which UART this function applies to.
- **enable** The allowed values are:

- 0 to disable the functionalities of the UART driver.
- 1 to enable the functionalities of the UART driver.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_INTERNAL_ERR	in case of internal problems with the hardware..

14.2.3 UAF_SetComPar

```
T_FDRET UAF_SetComPar ( T_fd_UartId uartNo,
                        T_baudrate baudrate,
                        T_bitsPerCharacter bpc,
                        T_stopBits sb,
                        T_parity parity )
```

Description

This function sets up the communication parameters of the Fax & Data serial data flow: used **baudrate**, **bpc** data bits and **sb** stop bits per character, and used **parity**.

Parameters

- **uartNo**

This parameter indicates which UART this function applies to.

- **baudrate**

This parameter indicates the used baud rate for the serial connection.

- **Bpc**

This parameter corresponds to the number of used data bits per character.

- **Sb**

This parameter indicates the number of used stop bits per character.

- **Parity**

This parameter corresponds to the used parity for the serial connection.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition

FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_INTERNAL_ERR	in case of internal problems with the hardware..

14.2.4 UAF_SetBuffer

```
T_FDRET UAF_SetBuffer ( T_fd_UartId uartNo,
                        SYS_UWORD16 bufSize,
                        SYS_UWORD16 rxThreshold,
                        SYS_UWORD16 txThreshold )
```

Description

This function sets up the size of the circular buffers and the related thresholds of the UART driver. This function may be called only if the UART has been previously disabled with UAF_Enable.

Parameters

- **uartNo**

This parameter indicates which UART this function applies to.

- **bufSize**

This parameter indicates the size of the Rx and Tx circular buffers.

- **RxThreshold**

This parameter corresponds to the amount of received bytes that leads to a call to the suspended read-out function, which is passed to the function UAF_ReadData.

- **TxThreshold**

This parameter corresponds to the amount of bytes in the Tx buffer that leads to a call to the suspended write-in function, which is passed to the function UAF_WriteData.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards or if the **bufSize** exceeds the maximal possible capabilities of the driver or the threshold values don't correspond to the **bufSize**,
FD_INTERNAL_ERR	in case of internal problems with the hardware or if the function has been called while the UART is enabled .

14.2.5 UAF_SetFlowCtrl

```
T_FDRET UAF_SetFlowCtrl ( T_fd_UartId uartNo,
                          T_flowCtrlMode fcMode,
                          SYS_UWORD8 XON,
                          SYS_UWORD8 XOFF )
```

Description

This function changes the flow control mode of the UART driver to **fcMode**. If no flow control is set, a loss of character should be prevented by the application. If a flow control is set, DTR or RTS is activated or XOFF is sent if the Rx buffer is not able to store the received characters. Otherwise DTR or RTS is deactivated or XON is sent.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

- **fcMode**

  This parameter indicates the used flow control mode for the serial connection.

- **XON**

This parameter corresponds to the ASCII code of the XON character. This parameter is ignored if **fcMode** is not set to fc_xoff.

- **XOFF**

This parameter corresponds to the ASCII code of the XOFF character. This parameter is ignored if **fcMode** is not set to fc_xoff.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards or if the selected flow control mode is not supported
FD_INTERNAL_ERR	in case of internal problems with the hardware.

14.2.6 UAF_SetEscape

```
T_FDRET UAF_SetEscape ( T_fd_UartId uartNo,
                        char escChar,
                        SYS_UWORD16 guardPeriod )
```

Description

To return to the command mode of the ACI while a data connection is established, an escape sequence (composed of three **escChar**) has to be detected. To distinguish between user data and the escape sequence a defined **guardPeriod** duration is necessary before and after this sequence. With this function the escape character and the guard period related to the UART driver can be set up.

Parameters

- **uartNo**

This parameter indicates which UART this function applies to.

- **escChar**

This parameter corresponds to the ASCII code of the character which could appear three times as an escape sequence.

- **guardPeriod**

This parameter denotes the minimal duration of the rest before the first and after the last **escChar** character of the escape sequence, and the maximal receiving duration of the whole escape string. This value is expressed in milliseconds (see document [3] for type definition).

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards or if the selected flow control mode is not supported
FD_INTERNAL_ERR	in case of internal problems with the hardware.

14.2.7 UAF_ReadData

```
T_FDRET UAF_ReadData ( T_fd_UartId uartNo,
                       T_suspendMode suspend,
                       void (readOutFunc ( SYS_BOOL cldFromIrq,
                                           T_reInstMode *reInstall,
                                           SYS_UWORD8 nsource,
                                           SYS_UWORD8 *source[],
                                           SYS_UWORD16 size[],
                                           SYS_UWORD32 state )))
```

Description

To read the received characters out of the Rx buffer the address of a function is passed. If characters are available, the driver calls this function and passes the source address and the amount of readable characters. Because the Rx buffer is circular, callback function may be called with more than one address of buffer fragment. The readOutFunc function modifies the contents of the size array to return the driver the number of processed characters. Each array entry is decremented by the number of bytes read in the fragment. If the function is called while the Rx buffer is empty, it depends on the suspend parameter to suspend the callback or to leave without any operation. In the case of suspension, the return value is FD_SUSPENDED (refer to §5.2). A delayed callback will be performed if:

- the Rx buffer reaches the adjusted threshold (rxThreshold of UAF_SetBuffer - refer to §6.4),
- the state of a V.24 input line has changed,
- a break is detected,

- an escape sequence is detected.

If no suspension is necessary the function returns the number of processed bytes.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

- **suspend**

This parameter indicates the mode of suspension in case of Rx buffer empty.

- **void (readOutFunc (SYS_BOOL cldFromIrq,**
  T_reInstMode *reInstall,
  SYS_UWORD8 nsource,
  SYS_UWORD8 *source[],
  SYS_UWORD16 size[],
  SYS_UWORD32 state)))

  This parameter corresponds to the callback function with all its parameters:

- CldFromIrq

Indicates if the callback function is called from an interrupt service routine (= TRUE = 1) or not (= FALSE = 0).

- reInstall

The callback function sets this parameter to rm_reInstall if the driver must call again the callback function when the Rx threshold level is reached. Else it will be set to rm_noInstall. Before to call the readOutFunc function this parameter is set to rm_notDefined.

- nsource

This parameter informs the callback function about the number of fragments which are ready to copy from the circular Rx buffer.

- source

Corresponding to nsource, this array contains the addresses of the fragments.

- size

Corresponding to nsource and source, this array contains the sizes of each fragments.

- state

This parameter corresponds to the status of the V.24 lines and the break / escape detection.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
Any values ≥ 0	which corresponds to a successful operation and especially to the amount of processed data
FD_NOT_SUPPORTED	if **uartNo** corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_SUSPENDED	when the callback is suspended until the buffer or

	state condition change,
FD_NOT_READY	if the function is called while the callback mechanism is activated and still not terminated,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

14.2.8 UAF_WriteData

```
T_FDRET UAF_WriteData (T_fd_UartId uartNo,
                       T_suspendMode suspend,
                       void (writeInFunc (SYS_BOOL cldFromIrq,
                                          T reInstMode *reInstall,
                                          SYS_UWORD8 ndest,
                                          SYS_UWORD8 *dest [],
                                          SYS_UWORD16 size [])))
```

Description

To write characters into the Tx buffer the address of a function is passed. If free space is available in the buffer, the driver calls this function and passes the destination address and the amount of space. Because the Tx buffer is circular, the callback function may be called with more than one address of buffer fragment. The writeInFunc function modifies the contents of the size array to return the driver the number of processed bytes. Each array entry is decremented by the number of bytes written in this fragment. If the function is called while the Tx buffer is full, it depends on the suspend parameter to suspend the callback or to leave this function without any operation. In the case of suspension the returned value is FD_SUSPENDED . A delayed callback will be performed if the Tx buffer reaches the adjusted threshold . If no suspension is necessary the function returns the number of processed bytes.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

- **suspend**

This parameter indicates the mode of suspension in case of Tx buffer empty (see §5.8 for allowed values).

- **void (writeInFunc (SYS_BOOL cldFromIrq,**
  T_reInstMode *reInstall,
  SYS_UWORD8 ndest,
  SYS_UWORD8 *dest[],
  SYS_UWORD16 size[])))

  This parameter corresponds to the callback function with all its parameters

- CldFromIrq

Indicates if the callback function is called from an interrupt service routine (= TRUE = 1) or not (= FALSE = 0).

- reInstall

The callback function sets this parameter to rm_reInstall if the driver must call again the callback function when the Tx threshold level is reached. Else it will be set to rm_noInstall.Before to call the writeInFunc function this parameter is set to rm_notDefined.

- Ndest

This parameter informs the callback function about the number of fragments which are available in the Tx buffer.

- dest

Corresponding to ndest, this array contains the addresses of the fragments.

- Size

Corresponding to ndest and dest, this array contains the sizes of each fragments.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
Any values ≥ 0	which corresponds to a successful operation and especially to the amount of processed data
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_SUSPENDED	when the callback is suspended until the buffer or state condition change,
FD_NOT_READY	if the function is called while the callback mechanism is activated and still not terminated,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

14.2.9 UAF_InpAvail

```
T_FDRET UAF_InpAvail ( T_fd_UartId uartNo )
```

Description

This function allows knowing the number of available characters in the Rx circular buffer of the UART driver.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
Any values > 0	Which correspond to the amount of data in the Rx buffer.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
0	if the function is called while the driver is

	disabled,
FD_NOT_READY	if the function is called while the callback mechanism is activated and still not terminated,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

14.2.10 UAF_OutpAvail

T_FDRET UAF_OutpAvail (T_fd_UartId uartNo)

Description

This function allows knowing the number of characters in the Tx circular buffer of the UART driver.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
Any values > 0	which correspond to the amount of data in the Tx buffer.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
0	if the function is called while the driver is disabled,
FD_NOT_READY	if the function is called while the callback mechanism is activated and still not terminated,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

14.2.11 UAF_EnterSleep

T_FDRET UAF_EnterSleep (T_fd_UartId uartNo)

Description

This function determines if the UART device is ready to enter deep sleep mode or not. Moreover, if the UART is ready, it is set up so that it may be waked up by the dedicated wake-up interrupt.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards,
0	(= FALSE) when the UART device is not ready to enter deep sleep mode
1	(= TRUE) when the UART is ready to enter deep sleep mode and has also been set up in order to be waked-up

14.2.12 UAF_WakeUp

```
T_FDRET UAF_WakeUp ( T_fd_UartId uartNo )
```

Description

When called, this function wakes up the UART device: wake-up interrupt disabled and usual Rx, Tx and Modem interrupts again unmasked.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_OK	which corresponds to a successful operation.

14.2.13 UAF_StopRec

```
T_FDRET UAF_StopRec ( T_fd_UartId uartNo )
```

Description

If a flow control mode is set, this function tells the terminal equipment (e.g. a PC) that no more data can be received by the UART driver: DTR or RTS is deactivated, or XOFF is sent. If this function is called, only a call to UAF_StartRec or an initialization restarts the transmission from the terminal equipment.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation,
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

14.2.14 UAF_StartRec

```
T_FDRET UAF_StartRec ( T_fd_UartId uartNo )
```

Description

If a flow control mode is set, this function tells the terminal equipment (e.g. a PC) that the receiver of the UART driver is able to receive some data: DTR or RTS is activated, or XON is sent.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation,
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

14.2.15 UAF_GetLineState

```
T_FDRET UAF_GetLineState ( T_fd_UartId uartNo,SYS_UWORD32 *state )
```

Description

This function returns the state of the V.24 lines, the flow control state and the result of the break / escape detection process as a bit field, related to the Fax & Data serial data flow.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

- **state**

This parameter represents the state of the V.24 lines, the flow control state and the result of the break / escape sequence detection process as a bit field (see document [3] for type definition).

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation,
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards
FD_NOT_READY	if the function is called while the callback mechanism of the read-out function is activated and still not terminated,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

14.2.16 UAF_SetLineState

```
T_FDRET UAF_SetLineState ( T_fd_UartId uartNo,
                           SYS_UWORD32 state,
                           SYS_UWORD32 mask )
```

Description

This function sets the states of the V.24 status lines according to the bits fields of the parameters state and mask.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

- **state**

  This parameter corresponds to the bits field used to change the states of the V.24 status lines.

- **mask**

This parameter corresponds to the mask used to manipulate the bits of the states of the V.24 status lines.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards
FD_INTERNAL_ERR	in case of internal problems with the hardware.

14.2.17 UAF_CheckXEmpty

```
T_FDRET UAF_CheckXEmpty ( T_fd_UartId uartNo )
```

Description

This function checks the empty condition of the transmitter of the UART driver: hardware FIFO and software buffer both empty and last character sent.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which means that the empty condition is OK,
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards
FD_NOT_READY	indicates that the transmitter is not empty,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

14.3 Types definition

14.3.1 T_fd_UartId

The purpose of this type is to define the different UARTs available on Hercules chipset (see document [2] for more details):

```
typedef enum {
UAF_UART_0,
UAF_UART_1,
UAF_UART_2
} T_fd_UartId;
```

UAF_UART_0: This value corresponds to the UART IrDA,
UAF_UART_1: This value corresponds to the UART Modem.
UAF_UART_2: This value corresponds to the UART Modem2 available on E-Sample platform only.

Note: The current implementation of IrDA on TI Development Boards B-Sample, C-Sample, D-Sample and E-Sample does not support hardware flow control (either CTS/RTS or DCD/DTR) and thus can't be used for Fax & Data. Moreover on E-Sample platform, Modem2 can't be use for Fax & Data since this UART device is not accessible through a DB9 connector.

14.3.2 T_FDRET

The purpose of this type is to define the returned values of each UAF_XXX functions. This type only re-defines the signed short integers (16 bits). The allowed values vary from **- 32 768** till **+ 32 767**:

```
typedef short T_FDRET;
```

Some specific values of this type are defined through user constants:

FD_OK (= 0): Value usually returned by a function when the performed operation is successful.
FD_SUSPENDED (= -1): Value usually returned by a function to indicate that the performed operation has been suspended.
FD_NOT_SUPPORTED (= -2): Value usually returned by a function when its input parameters don't fit to the hardware or driver capabilities.
FD_NOT_READY (= -3): Value usually returned by a function called while the hardware or the driver was not ready to perform the requested operation.
FD_INTERNAL_ERR (= -9): Value usually returned by a function in case of internal problems with the hardware.

This type definition is within the file "faxdata.h".

14.3.3 T_baudrate

The purpose of this type is to define the various baudrates available on the UART device:

```
typedef enum {
                FD_BAUD_AUTO,
                FD_BAUD_75,
                FD_BAUD_150,
                FD_BAUD_300,
                FD_BAUD_600,
                FD_BAUD_1200,
                FD_BAUD_2400,
                FD_BAUD_4800,
                FD_BAUD_7200,
                FD_BAUD_9600,
                FD_BAUD_14400,
                FD_BAUD_19200,
```

```
        FD_BAUD_28800,
        FD_BAUD_33900,
        FD_BAUD_38400,
        FD_BAUD_57600,
        FD_BAUD_115200,
        FD_BAUD_203125,
        FD_BAUD_406250,
        FD_BAUD_812500
        } T_baudrate;
```

This type definition is within the file “faxdata.h”.

14.3.4 T_bitsPerCharacter

The purpose of this type is to describe the number of used data bits per character:

```
typedef enum {
        bpc_7,
        bpc_8
        } T_bitsPerCharacter;
```

This type definition is within the file “faxdata.h”.

14.3.5 T_stopBits

The purpose of this type is to describe the number of used stop bits per character:

```
typedef enum {
        sb_1,
        sb_2
        } T_stopBits;
```

This type definition is within the file “faxdata.h”.

14.3.6 T_parity

The purpose of this type is to describe the used parity for the serial connection:

```
typedef enum {
        pa_none,
        pa_even,
        pa_odd,
        pa_space
        } T_parity;
```

This type definition is within the file “faxdata.h”.

14.3.7 T_flowCtrlMode

The purpose of this type is to describe the used flow control mode for the serial connection:

```
typedef enum {
        fc_none,
        fc_rts,
```

```
fc_dtr,
fc_xoff
} T_flowCtrlMode;
```

This type definition is within the file “faxdata.h”.

14.3.8 T_suspendMode

The purpose of this type is to describe the mode of suspension when a callback mechanism is activated while the related buffer of the driver (Rx or Tx) is empty.

```
typedef enum {
        sm_noSuspend,
        sm_suspend
        } T_suspendMode;
```

This type definition is within the file “faxdata.h”.

14.3.9 T_reInstMode

The purpose of this type is to describe the install mode of the callback function.

```
typedef enum {
        rm_notDefined,
        rm_reInstall,
        rm_noInstall
        } T_reInstMode;
```

This type definition is within the file “faxdata.h”.

Chapter 15 UART

Error! Reference source not found. Error! Reference source not found. Error! Bookmark not defined.

Error! Reference source not found. Interface Definition **Error! Bookmark not defined.**

Error! Reference source not found. Error! Reference source not found. Error! Bookmark not defined.

15.1 Introduction

The purpose of this document is to describe the interface of the functions provided by the UART driver, when the UART device is used for Fax & Data.

This driver only allows managing the UART Modem serial device of the B-Sample, C-Sample, D-Sample and E-Sample Texas Instruments Development Boards. Indeed, the UART IrDA device of these boards can't be used for Fax & Data because the hardware flow control (either CTS/RTS for B-Sample or DCD/DTR for C-Sample, D-Sample and E-Sample) is not supported.
Moreover the UART Modem2 of the E-Sample platform cannot be used for Fax & Data because this device is not directly accessible through a DB9 connector.

The use of serial ports of Texas Instruments Development Boards (B-Sample, C-Sample, D-Sample or E-Sample) is directly managed by a software module called serialswitch .

A software application (GSM/GPRS Protocol Stack) deals with several serial data flows that are dynamically connected by the serialswitch module to the hardware serial devices through software UART drivers.

The figure below illustrates how the serial data flows of a GSM/GPRS Protocol Stack application could be linked to the hardware serial devices.

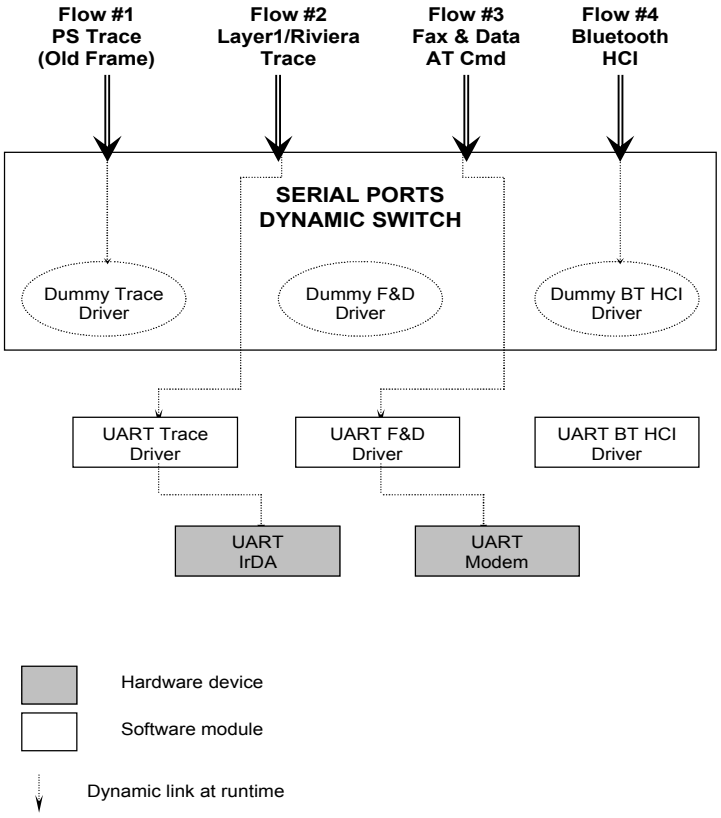


Figure 11 Data flows and HW devices

15.2 Interface description

15.2.1 UAF_Init

```
T_FDRET UAF_Init (T_fd_UartId uartNo)
```

Description

This function initializes the UART hardware used for Fax & Data, and installs the interrupt handlers. The parameters are set to the following default values:

- 19200 bps,
- 8 data bits per character,
- 1 stop bit,
- no parity,
- no flow control.

All functionalities of the UART driver are disabled, and must be enabled by calling the function UAF_Enable.

Parameters

- **uartNo** This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_INTERNAL_ERR	in case of internal problems with the hardware..

15.2.2 UAF_Enable

```
T_FDRET UAF_Enable(T_fd_uartid uartNo,SYS_BOOL enable)
```

Description

This function enables or disables the functionalities of the UART driver used for Fax & Data, according to the value of **enable**. In the deactivated state, all information about the communication parameters are stored and recalled if the driver is again enabled. When the driver is enabled the Rx and Tx buffers are cleared.

Parameters

- **uartNo** This parameter indicates which UART this function applies to.
- **enable** The allowed values are:

- 0 to disable the functionalities of the UART driver.
- 1 to enable the functionalities of the UART driver.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_INTERNAL_ERR	in case of internal problems with the hardware..

15.2.3 UAF_SetComPar

```
T_FDRET UAF_SetComPar ( T_fd_UartId uartNo,
                        T_baudrate baudrate,
                        T_bitsPerCharacter bpc,
                        T_stopBits sb,
                        T_parity parity )
```

Description

This function sets up the communication parameters of the Fax & Data serial data flow: used **baudrate**, **bpc** data bits and **sb** stop bits per character, and used **parity**.

Parameters

- **uartNo**

This parameter indicates which UART this function applies to.

- **baudrate**

This parameter indicates the used baud rate for the serial connection.

- **Bpc**

This parameter corresponds to the number of used data bits per character.

- **Sb**

This parameter indicates the number of used stop bits per character.

- **Parity**

This parameter corresponds to the used parity for the serial connection.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition

FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_INTERNAL_ERR	in case of internal problems with the hardware..

15.2.4 UAF_SetBuffer

```
T_FDRET UAF_SetBuffer ( T_fd_UartId uartNo,
                        SYS_UWORD16 bufSize,
                        SYS_UWORD16 rxThreshold,
                        SYS_UWORD16 txThreshold )
```

Description

This function sets up the size of the circular buffers and the related thresholds of the UART driver. This function may be called only if the UART has been previously disabled with UAF_Enable.

Parameters

- **uartNo**

This parameter indicates which UART this function applies to.

- **bufSize**

This parameter indicates the size of the Rx and Tx circular buffers.

- **RxThreshold**

This parameter corresponds to the amount of received bytes that leads to a call to the suspended read-out function, which is passed to the function UAF_ReadData.

- **TxThreshold**

This parameter corresponds to the amount of bytes in the Tx buffer that leads to a call to the suspended write-in function, which is passed to the function UAF_WriteData.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards or if the **bufSize** exceeds the maximal possible capabilities of the driver or the threshold values don't correspond to the **bufSize**,
FD_INTERNAL_ERR	in case of internal problems with the hardware or if the function has been called while the UART is enabled .

15.2.5 UAF_SetFlowCtrl

```
T_FDRET UAF_SetFlowCtrl ( T_fd_UartId uartNo,
                          T_flowCtrlMode fcMode,
                          SYS_UWORD8 XON,
                          SYS_UWORD8 XOFF )
```

Description

This function changes the flow control mode of the UART driver to **fcMode**. If no flow control is set, a loss of character should be prevented by the application. If a flow control is set, DTR or RTS is activated or XOFF is sent if the Rx buffer is not able to store the received characters. Otherwise DTR or RTS is deactivated or XON is sent.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

- **fcMode**

  This parameter indicates the used flow control mode for the serial connection.

- **XON**

This parameter corresponds to the ASCII code of the XON character. This parameter is ignored if **fcMode** is not set to fc_xoff.

- **XOFF**

This parameter corresponds to the ASCII code of the XOFF character. This parameter is ignored if **fcMode** is not set to fc_xoff.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards or if the selected flow control mode is not supported
FD_INTERNAL_ERR	in case of internal problems with the hardware.

15.2.6 UAF_SetEscape

```
T_FDRET UAF_SetEscape ( T_fd_UartId uartNo,
                        char escChar,
                        SYS_UWORD16 guardPeriod )
```

Description

To return to the command mode of the ACI while a data connection is established, an escape sequence (composed of three **escChar**) has to be detected. To distinguish between user data and the escape sequence a defined **guardPeriod** duration is necessary before and after this sequence. With this function the escape character and the guard period related to the UART driver can be set up.

Parameters

- **uartNo**

This parameter indicates which UART this function applies to.

- **escChar**

This parameter corresponds to the ASCII code of the character which could appear three times as an escape sequence.

- **guardPeriod**

This parameter denotes the minimal duration of the rest before the first and after the last **escChar** character of the escape sequence, and the maximal receiving duration of the whole escape string. This value is expressed in milliseconds (see document [3] for type definition).

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards or if the selected flow control mode is not supported
FD_INTERNAL_ERR	in case of internal problems with the hardware.

15.2.7 UAF_ReadData

```
T_FDRET UAF_ReadData ( T_fd_UartId uartNo,
                       T_suspendMode suspend,
                       void (readOutFunc ( SYS_BOOL cldFromIrq,
                                           T_reInstMode *reInstall,
                                           SYS_UWORD8 nsource,
                                           SYS_UWORD8 *source[],
                                           SYS_UWORD16 size[],
                                           SYS_UWORD32 state )))
```

Description

To read the received characters out of the Rx buffer the address of a function is passed. If characters are available, the driver calls this function and passes the source address and the amount of readable characters. Because the Rx buffer is circular, callback function may be called with more than one address of buffer fragment. The readOutFunc function modifies the contents of the size array to return the driver the number of processed characters. Each array entry is decremented by the number of bytes read in the fragment. If the function is called while the Rx buffer is empty, it depends on the suspend parameter to suspend the callback or to leave without any operation. In the case of suspension, the return value is FD_SUSPENDED (refer to §5.2). A delayed callback will be performed if:

- the Rx buffer reaches the adjusted threshold (rxThreshold of UAF_SetBuffer - refer to §6.4),
- the state of a V.24 input line has changed,
- a break is detected,

- an escape sequence is detected.

If no suspension is necessary the function returns the number of processed bytes.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

- **suspend**

This parameter indicates the mode of suspension in case of Rx buffer empty.

- **void (readOutFunc (SYS_BOOL cldFromIrq,**
  T_reInstMode *reInstall,
  SYS_UWORD8 nsource,
  SYS_UWORD8 *source[],
  SYS_UWORD16 size[],
  SYS_UWORD32 state)))

  This parameter corresponds to the callback function with all its parameters:

- CldFromIrq

Indicates if the callback function is called from an interrupt service routine (= TRUE = 1) or not (= FALSE = 0).

- reInstall

The callback function sets this parameter to rm_reInstall if the driver must call again the callback function when the Rx threshold level is reached. Else it will be set to rm_noInstall. Before to call the readOutFunc function this parameter is set to rm_notDefined.

- nsource

This parameter informs the callback function about the number of fragments which are ready to copy from the circular Rx buffer.

- source

Corresponding to nsource, this array contains the addresses of the fragments.

- size

Corresponding to nsource and source, this array contains the sizes of each fragments.

- state

This parameter corresponds to the status of the V.24 lines and the break / escape detection.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
Any values ≥ 0	which corresponds to a successful operation and especially to the amount of processed data
FD_NOT_SUPPORTED	if **uartNo** corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_SUSPENDED	when the callback is suspended until the buffer or

	state condition change,
FD_NOT_READY	if the function is called while the callback mechanism is activated and still not terminated,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

15.2.8 UAF_WriteData

```
T_FDRET UAF_WriteData (T_fd_UartId uartNo,
                       T_suspendMode suspend,
                       void (writeInFunc (SYS_BOOL cldFromIrq,
                                          T reInstMode *reInstall,
                                          SYS_UWORD8 ndest,
                                          SYS_UWORD8 *dest [],
                                          SYS_UWORD16 size [])))
```

Description

To write characters into the Tx buffer the address of a function is passed. If free space is available in the buffer, the driver calls this function and passes the destination address and the amount of space. Because the Tx buffer is circular, the callback function may be called with more than one address of buffer fragment. The writeInFunc function modifies the contents of the size array to return the driver the number of processed bytes. Each array entry is decremented by the number of bytes written in this fragment. If the function is called while the Tx buffer is full, it depends on the suspend parameter to suspend the callback or to leave this function without any operation. In the case of suspension the returned value is FD_SUSPENDED . A delayed callback will be performed if the Tx buffer reaches the adjusted threshold . If no suspension is necessary the function returns the number of processed bytes.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

- **suspend**

This parameter indicates the mode of suspension in case of Tx buffer empty (see §5.8 for allowed values).

- **void (writeInFunc (SYS_BOOL cldFromIrq,**
  T_reInstMode *reInstall,
  SYS_UWORD8 ndest,
  SYS_UWORD8 *dest[],
  SYS_UWORD16 size[])))

  This parameter corresponds to the callback function with all its parameters

- CldFromIrq

Indicates if the callback function is called from an interrupt service routine (= TRUE = 1) or not (= FALSE = 0).

- reInstall

The callback function sets this parameter to rm_reInstall if the driver must call again the callback function when the Tx threshold level is reached. Else it will be set to rm_noInstall.Before to call the writeInFunc function this parameter is set to rm_notDefined.

- Ndest

This parameter informs the callback function about the number of fragments which are available in the Tx buffer.

- dest

Corresponding to ndest, this array contains the addresses of the fragments.

- Size

Corresponding to ndest and dest, this array contains the sizes of each fragments.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
Any values ≥ 0	which corresponds to a successful operation and especially to the amount of processed data
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_SUSPENDED	when the callback is suspended until the buffer or state condition change,
FD_NOT_READY	if the function is called while the callback mechanism is activated and still not terminated,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

15.2.9 UAF_InpAvail

```
T_FDRET UAF_InpAvail ( T_fd_UartId uartNo )
```

Description

This function allows knowing the number of available characters in the Rx circular buffer of the UART driver.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
Any values > 0	Which correspond to the amount of data in the Rx buffer.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
0	if the function is called while the driver is

	disabled,
FD_NOT_READY	if the function is called while the callback mechanism is activated and still not terminated,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

15.2.10 UAF_OutpAvail

T_FDRET UAF_OutpAvail (T_fd_UartId uartNo)

Description

This function allows knowing the number of characters in the Tx circular buffer of the UART driver.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
Any values > 0	which correspond to the amount of data in the Tx buffer.
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
0	if the function is called while the driver is disabled,
FD_NOT_READY	if the function is called while the callback mechanism is activated and still not terminated,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

15.2.11 UAF_EnterSleep

T_FDRET UAF_EnterSleep (T_fd_UartId uartNo)

Description

This function determines if the UART device is ready to enter deep sleep mode or not. Moreover, if the UART is ready, it is set up so that it may be waked up by the dedicated wake-up interrupt.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards,
0	(= FALSE) when the UART device is not ready to enter deep sleep mode
1	(= TRUE) when the UART is ready to enter deep sleep mode and has also been set up in order to be waked-up

15.2.12 UAF_WakeUp

```
T_FDRET UAF_WakeUp ( T_fd_UartId uartNo )
```

Description

When called, this function wakes up the UART device: wake-up interrupt disabled and usual Rx, Tx and Modem interrupts again unmasked.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 which can't be used for Fax & Data on TI Development Boards,
FD_OK	which corresponds to a successful operation.

15.2.13 UAF_StopRec

```
T_FDRET UAF_StopRec ( T_fd_UartId uartNo )
```

Description

If a flow control mode is set, this function tells the terminal equipment (e.g. a PC) that no more data can be received by the UART driver: DTR or RTS is deactivated, or XOFF is sent. If this function is called, only a call to UAF_StartRec or an initialization restarts the transmission from the terminal equipment.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation,
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

15.2.14 UAF_StartRec

```
T_FDRET UAF_StartRec ( T_fd_UartId uartNo )
```

Description

If a flow control mode is set, this function tells the terminal equipment (e.g. a PC) that the receiver of the UART driver is able to receive some data: DTR or RTS is activated, or XON is sent.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation,
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

15.2.15 UAF_GetLineState

```
T_FDRET UAF_GetLineState ( T_fd_UartId uartNo,SYS_UWORD32 *state )
```

Description

This function returns the state of the V.24 lines, the flow control state and the result of the break / escape detection process as a bit field, related to the Fax & Data serial data flow.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

- **state**

This parameter represents the state of the V.24 lines, the flow control state and the result of the break / escape sequence detection process as a bit field (see document [3] for type definition).

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation,
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards
FD_NOT_READY	if the function is called while the callback mechanism of the read-out function is activated and still not terminated,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

15.2.16 UAF_SetLineState

```
T_FDRET UAF_SetLineState ( T_fd_UartId uartNo,
                           SYS_UWORD32 state,
                           SYS_UWORD32 mask )
```

Description

This function sets the states of the V.24 status lines according to the bits fields of the parameters state and mask.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

- **state**

  This parameter corresponds to the bits field used to change the states of the V.24 status lines.

- **mask**

This parameter corresponds to the mask used to manipulate the bits of the states of the V.24 status lines.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which corresponds to a successful operation
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards
FD_INTERNAL_ERR	in case of internal problems with the hardware.

15.2.17 UAF_CheckXEmpty

```
T_FDRET UAF_CheckXEmpty ( T_fd_UartId uartNo )
```

Description

This function checks the empty condition of the transmitter of the UART driver: hardware FIFO and software buffer both empty and last character sent.

Parameters

- **uartNo**

  This parameter indicates which UART this function applies to.

Immediate Return

- **T_FDRET**

The function returns immediate with the following possible values:

Id	Definition
FD_OK	which means that the empty condition is OK,
FD_NOT_SUPPORTED	if uartNo corresponds to the UART IrDA or UART Modem2 (UAF_UART_0 or UAF_UART_2 - refer to §5.1) which can't be used for Fax & Data on TI Development Boards
FD_NOT_READY	indicates that the transmitter is not empty,
FD_INTERNAL_ERR	in case of internal problems with the hardware.

15.3 Types definition

15.3.1 T_fd_UartId

The purpose of this type is to define the different UARTs available on Hercules chipset (see document [2] for more details):

```
typedef enum {
UAF_UART_0,
UAF_UART_1,
UAF_UART_2
} T_fd_UartId;
```

UAF_UART_0: This value corresponds to the UART IrDA,
UAF_UART_1: This value corresponds to the UART Modem.
UAF_UART_2: This value corresponds to the UART Modem2 available on E-Sample platform only.

Note: The current implementation of IrDA on TI Development Boards B-Sample, C-Sample, D-Sample and E-Sample does not support hardware flow control (either CTS/RTS or DCD/DTR) and thus can't be used for Fax & Data. Moreover on E-Sample platform, Modem2 can't be use for Fax & Data since this UART device is not accessible through a DB9 connector.

15.3.2 T_FDRET

The purpose of this type is to define the returned values of each UAF_XXX functions. This type only re-defines the signed short integers (16 bits). The allowed values vary from **- 32 768** till **+ 32 767**:

```
typedef short T_FDRET;
```

Some specific values of this type are defined through user constants:

FD_OK (= 0): Value usually returned by a function when the performed operation is successful.
FD_SUSPENDED (= -1): Value usually returned by a function to indicate that the performed operation has been suspended.
FD_NOT_SUPPORTED (= -2): Value usually returned by a function when its input parameters don't fit to the hardware or driver capabilities.
FD_NOT_READY (= -3): Value usually returned by a function called while the hardware or the driver was not ready to perform the requested operation.
FD_INTERNAL_ERR (= -9): Value usually returned by a function in case of internal problems with the hardware.

This type definition is within the file "faxdata.h".

15.3.3 T_baudrate

The purpose of this type is to define the various baudrates available on the UART device:

```
typedef enum {
                FD_BAUD_AUTO,
                FD_BAUD_75,
                FD_BAUD_150,
                FD_BAUD_300,
                FD_BAUD_600,
                FD_BAUD_1200,
                FD_BAUD_2400,
                FD_BAUD_4800,
                FD_BAUD_7200,
                FD_BAUD_9600,
                FD_BAUD_14400,
                FD_BAUD_19200,
```

```
        FD_BAUD_28800,
        FD_BAUD_33900,
        FD_BAUD_38400,
        FD_BAUD_57600,
        FD_BAUD_115200,
        FD_BAUD_203125,
        FD_BAUD_406250,
        FD_BAUD_812500
        } T_baudrate;
```

This type definition is within the file “faxdata.h”.

15.3.4 T_bitsPerCharacter

The purpose of this type is to describe the number of used data bits per character:

```
typedef enum {
        bpc_7,
        bpc_8
        } T_bitsPerCharacter;
```

This type definition is within the file “faxdata.h”.

15.3.5 T_stopBits

The purpose of this type is to describe the number of used stop bits per character:

```
typedef enum {
        sb_1,
        sb_2
        } T_stopBits;
```

This type definition is within the file “faxdata.h”.

15.3.6 T_parity

The purpose of this type is to describe the used parity for the serial connection:

```
typedef enum {
        pa_none,
        pa_even,
        pa_odd,
        pa_space
        } T_parity;
```

This type definition is within the file “faxdata.h”.

15.3.7 T_flowCtrlMode

The purpose of this type is to describe the used flow control mode for the serial connection:

```
typedef enum {
        fc_none,
        fc_rts,
```

```
fc_dtr,
fc_xoff
} T_flowCtrlMode;
```

This type definition is within the file “faxdata.h”.

15.3.8 T_suspendMode

The purpose of this type is to describe the mode of suspension when a callback mechanism is activated while the related buffer of the driver (Rx or Tx) is empty.

```
typedef enum {
        sm_noSuspend,
        sm_suspend
        } T_suspendMode;
```

This type definition is within the file “faxdata.h”.

15.3.9 T_reInstMode

The purpose of this type is to describe the install mode of the callback function.

```
typedef enum {
        rm_notDefined,
        rm_reInstall,
        rm_noInstall
        } T_reInstMode;
```

This type definition is within the file “faxdata.h”.

Chapter 16 I2C

16.1 Introduction

This document describes the interface (API) of the I2C driver . This driver provides high level services for communication with I2C devices which are connected to the I2C controller of the Locosto processor. The API of the I2C driver is a REMU type interface which is re-entrant and non-blocking.

The driver supports Multi-master transmitter/receiver communication mode but does not support Slave receiver/transmitter communication mode. This means that the Locosto will always be master.

The driver support 7-bit and 10-bit addressing mode which is configurable at compile time. The driver support standard mode (up to 100 Kbits/s) and fast mode (up to 400 Kbits/s) which is configurable at compile time.

The detailed requirements of the I2C-SWE can be found in **Error! Reference source not found.**. Specification of the I2C hardware sub-system can be found in **Error! Reference source not found.**. Information about the I^2C bus standard can be found in **Error! Reference source not found.**.

(Provide a short description of this document. Describe which driver is documented, the purpose of the driver, the services it provides, the scope of the driver, who will use the driver, which entities this driver is using (linked entities), etc.

16.2 Service functions definition

16.2.1 i2c_set_transfer_mode

```
T_RV_RET i2c_set_transfer_mode (E_I2C_TRANFER_MODE i2c_mode)
```

Description

This function selects the Transfer mode to be used by the driver. Default setting will be I2C_INTERUPT.

Parameters

- I2c_mode

Indicates whether to use interrupt-, or polling- base method to transfer data between the i2c controllers to the application.
I2C_INTERRUPT (Transfer data to the client ENTITY interrupt based)
I2C_POLLING (Transfer data to the client ENTITY polling based)
For interrupt and polling based transfer see also [I2C_HW_SPEC].

Immediate Return

- **T_RV_RET**

The function returns immediate with the following possible values:

Id	Definition
RV_OK	The API function was successfully executed.
RV_NOT_READY	The driver is not able to handle this request at this moment (SWE not initialised).
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.

Event Return

I2C_TRANSFER_MODE_RSP_MSG event is returned to the calling ENTITY in case the pointer to the call-back function in the return path contains NULL, if the pointer to the calback function is not NULL, the message will be returned as parameter of the callbackfunction.

Current restriction of use

None.

16.2.2 i2c_read

```
T_I2C_RETURN i2c_read (UINT16 address, UINT16 *read_buffer_p,UINT16
nmb_of_bytes, T_I2C_ENDIAN endian, T_RV_RETURN return_path);
```

Description

This function reads a number of bytes from an i2c address.

Parameters

- address

I2c address where the data is read from.

- read_buffer_p

Buffer where the readdata must be placed. (The client must allocate this buffer)

- nmb_of_bytes

Number of bytes to read from the i2c address

- endian

Indicates if Big endian or little endian is used.
Big endian first the high byte then low byte
Little endian first the low byte then high byte.

- return_path

The return path of the client. The structure provides information about the way the driver must react asynchronous (whether to use a call-back principle or a return message).

Immediate Return

- **T_RV_RET**

The function returns immediate with the following possible values:

Id	Definition
RV_OK	The API function was successfully executed.
RV_NOT_READY	The driver is not able to handle this request at this moment (SWE not initialised).
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.

Event Return

An I2C_READ_RSP_MSG event is returned to the calling ENTITY in case the pointer to the call-back function in the return path contains NULL, if the pointer to the calback function is not NULL, the message will be returned as parameter of the callbackfunction.

Current restriction of use

None.

16.2.3 i2c_write

```
T_I2C_RETURN i2c_write (UINT16 address, UINT16* write_buffer_p, UINT16
nmb_of_bytes, T_I2C_ENDIAN endian, T_RV_RETURN return_path);
```

Description

This function writes a number of bytes to an i2c address.

Parameters

- address

I2c address of the device, where the data is written to

- write_buffer_p

Pointer to the "buffer which contains the data that must be written. (The client must allocate this buffer)

- nmb_of_bytes

Number of bytes to write to the i2c address

- Endian

Indicates if Big endian or little endian is used.
Big endian first the high byte then low byte
Little endian first the low byte then high byte.

- return_path

The return path of the client. The structure provides information about the way the driver must react asynchronous (whether to use a call-back principle or a return message).

Immediate Return

- **T_RV_RET**

The function returns immediate with the following possible values:

id	Definition
RV_OK	The API function was successfully executed.
RV_NOT_READY	The driver is not able to handle this request at this moment (SWE not initialised).
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.

Event Return

An I2C_WRITE_RSP_MSG event is returned to the calling ENTITY in case the pointer to the call-back function in the return path contains NULL, if the pointer to the calback function is not NULL, the message will be returned as parameter of the callbackfunction.

Current restriction of use

None.

16.2.4 i2c_get_sw_version

```
UINT32 i2c_get_sw_version(void)
```

Description

This function returns the driver version.

Parameters

None.

Immediate Return

- **UINT32**

Bit	Name	Function
[0-15]	BUILD	Build number
[16-23]	MINOR	Minor version number
[24-31]	MAJOR	Major version number

Current restriction of use

None.

16.2.5 i2c_get_hw_version

```
UINT8 i2c_get_hw_version(void)
```

Description

This function returns the hardware version of the I2C device.

Parameters

None.

Immediate Return

- **UINT8**

Bit	Name	Function
[0-3]	MINOR	Minor version number
[4-7]	MAJOR	Major version number

Current restriction of use

None.

16.3 Test functions definition

16.3.1 i2c_set_system_test

```
T_RV_RET i2c_set_system_test (T_I2C_FREERUN     freerun_mode,
                              T_I2C_INT_SET     int_status,
                              T_I2C_TEST        test_mode,
                              UINT8             scl_value,
                              UINT8             sda_value)
```

Description

This function is used to facilitate system-level tests by overriding some of the standard functional features of the peripheral. It can permit the test of SCL counters, control the signals that connect to I/O pins for digital loop-back for self-test. It also provides stop/no-stop functionality in debug mode.

Caution: never use this register for normal I2C operation.

Parameters

- **freerun_mode**

With *freerun_mode* the behaviour of the I2C controller can be defined when a breakpoint is encountered in the debugger.

- **int_status**

With int_status the interrupt status bits as defined in T_I2C_INTERRUPT can all be set to 1.

- **test_mode**

With *test_mode* the device can put in two different test modes.

- The SCL counter test mode

Generate continuous clock signal on SCL pin. This is useful to test the prescaler and SCL high/low counters. (Verify speed settings). Parameters ***scl_value*** and ***sda_value*** has no meaning in this mode.

- Loopback mode.

The value of the ***scl_value*** and ***sda_value*** parameters are routed to the SCL and SDA lines. Use the function i2c_get_test_data to read back the effect on the output lines.
Possible open lines or shortcuts can be detected in this way.

- ***scl_value*** and ***sda_value***

Used only in combination with the parameter **test_mode** and only then when it selects the loopback test mode mode. Possible values for **scl_value** and **sda_value** are 1 for high and 0 for low.

Immediate Return

- **T_RV_RET**

The function returns immediate with the following possible values:

id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	The parameter is incorrect.
RV_NOT_READY	Requested process is supported but cannot be processed now (SWE not initialized).

Current restriction of use

None.

16.3.2 i2c_get_test_data

```
T_RV_RET i2c_get_test_data( UINT8   *scl_value_p,
                            UINT8   *sda_value_p)
```

Description

This function is used to read the status of the data and clock lines (SDA and SCL).
The returned values are only defined when the device is already set in the loopback system test mode (see **Error! Reference source not found.**).

Parameters

- ***scl_value_p*** and ***sda_val_p***

Must point to the location where the state of the SCL and SDA lines is to be written.
The returned values are only in valid when the system test mode is selected in the loopback .

Immediate Return

- **T_RV_RET**

The function returns immediate with the following possible values:

id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	The parameter is incorrect.
RV_NOT_READY	Requested process is supported but cannot be processed now (ENTITY not initialized).

Current restriction of use

None.

16.4 Message definition

16.4.1 I2C_TRANSFER_MODE_REQ_MSG

The I2C_TRANSFER_MODE_REQ_MSG can be used to set the transfer mode The driver does not respond

```
typedef struct {
        /** Message header. */
        T_RV_HDR                os_hdr;
        T_I2C_TRANFER_MODE      transfer_mode;
        T_RV_RETURN             return_path
        } T_I2C_TRANSFER_MODE_REQ_MSG;
```

16.4.2 I2C_READ_REQ_MSG

The I2C_READ_REQ_MSG message can be used to retrieve the data of an I2C address. The driver responds with a T_I2C_READ_RSP_MSG message.

```
typedef struct {
        T_RV_HDR        os_hdr;
        UINT16          address
        UINT8  *        read_buffer_p;
```

```
                UINT16          nmb_of_bytes;
                T_RV_RETURN     return_path;
                T_I2C_ENDIAN    endian
} T_I2C_READ_REQ_MSG;
```

16.4.3 I2C_READ_RSP_MSG

Response to the T_I2C_READ_REQ_MSG message

```
typedef struct {
                T_RV_HDR    os_hdr;
                T_RV_RET    result;
} T_ T_I2C_READ_RSP_MSG;
```

The possible values for 'result' are:

Id	Definition
RV_OK	The API function was successfully executed.
RV_NOT_READY	Requested process is supported but cannot be processed now (ENTITY not initialized).
RV_INVALID_PARAMETER	A parameter is out of it's valid range
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

16.4.4 I2C_WRITE_REQ_MSG

The I2C_WRITE_REQ_MSG message can be used to send the data to an I2C address. The driver responds with a T_I2C_WRITE_RSP_MSG message.

```
typedef struct {
T_RV_HDR            os_hdr;
                UINT16          address;
                UINT16          write_buffer_p;
                UINT16          nmb_of_bytes;
                T_I2C_ENDIAN    endian;
                T_RV_RETURN     return_path;
} T_I2C_READ_REQ_MSG;
```

16.4.5 I2C_WRITE_RSP_MSG

Response to the T_I2C_WRITE_REQ_MSG message

```
typedef struct {
                T_RV_HDR    os_hdr;
                T_RV_RET    result;
} T_ T_I2C_READ_RSP_MSG;
```

The possible values for 'result' are:

Id	Definition
RV_OK	The API function was successfully executed.
RV_NOT_READY	Requested process is supported but cannot be processed now (SWE not initialized).
RV_INVALID_PARAMETER	A parameter is out of it's valid range
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

16.5 Types definition

API type definitions are located in the configuration file i2c_api.h in the common directory.
None of the numbers used in definitions are to be changed by the client programmer!

16.5.1 T_I2C_TRANSFER_MODE

Defines the way the data is transferred to or from a calling ENTITY

T_I2c_TRANSFER_MODE can have the following values:

```
typedef enum  {
                I2C_INTERRUPT = 0,
                I2C_POLLING,
                I2C_DMA
                } T_I2C_TRANFER_MODE;
```

16.5.2 T_I2C_ENDIAN

Defines the way the data is transferred to or from a calling ENTITY

T_I2c_TRANSFER_MODE can have the following values:

```
typedef enum {
                I2C_LITTLE_ENDIAN = (0x000),
                I2C_BIG_ENDIAN = (0x4000)
} T_I2C_ENDIAN;
```

16.6 Configuration Items

#define I2C_PRESCALE_VALUE 0x01
The Locosto system clock is divided by this (value +1) to give the internal sampling clock in the i2c device

#define I2C_OWN_ADDRESS <value>
Possible values: 7 bit or 10 bit value depending on the selected addressing mode.

#define I2C_CLOCK_TIME_HIGH <value>
Possible values: ESAMPLE_100KHZ_HIGH, ESAMPLE_400KHZ_HIGH, user specific value.

#define I2C_CLOCK_TIME_LOW <value>
Possible values: ESAMPLE_100KHZ_LOW, ESAMPLE_400KHZ_LOW, user specific value.

#define I2C_ADRESS_MODE
Possible values: I2C_07_BITS_ADDRESS_MODE, I2C_10_BITS_ADDRESS_MODE.

16.7 ENTITY State diagram

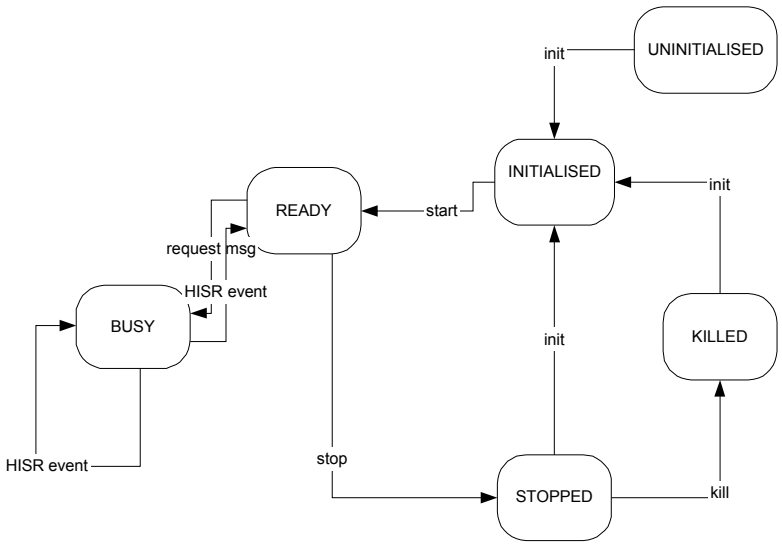


Figure 12 State machine behaviour

Chapter 17 KEYPAD

SUMMARY

Reference documents

Doc #	Document Title	Name
1	Riviera Manager Overview	RIV021 v0.5 (Texas Instruments)

Glossary

SWE	SoftWare Entity
MMI	Man Machine Interface

Introduction

This document provides an interface specification of the KEYPAD SW entity.

Overview

Generality

The main purpose of the keypad driver is to send messages to registered SWE when keys are pressed or released on the mobile.

Some notions used in the next paragraphs must be explained :

Client/server:
Keypad uses the client/server relationship. Indeed, the keypad is defined like a messages server for key pressed and keys released actions.
Numerous clients can subscribe to the keypad driver in order to receive these messages

Physical/virtual keys:
Two kinds of keys have been defined to break the link between user action on the physical key and the key message sent to the client.
Physical keys define the set of keys available on the physical keypad.
Virtual keys are just an abstraction of these physical keys in order to redefine the link between physical and virtual keys. One or more virtual keys can be linked to the same physical key.
This simplifies development and allows to change the keypad configuration without changing source code.
A client has no visibility on the physical keys. It only use virtual key ID for all the keypad services.

Mode:
A mode defines the link between physical and virtual keys. Two pre-defined mode exist, the first is the default mode, the second is the alphanumeric mode. Additional mode can be added when keypad driver is implemented according to the customer need.

The keypad driver allows to:

- Subscribe several SWE's for a predefined set of key. When one of these is pressed or released, SWE's and/or modules concerned are automatically informed.
- Unsubscribe from keypad (for all keys),
- Define repetition for a key (long press and repetition available for a specific subscriber)
- Define a keypad owner. subscriber is the only notified from key pressed or released until unsubscribe or cancel this privilege.
- Many definable mode (and one dynamic mode (for game configuration)) for associating physical key and functional key,
- Dynamically change mode,
- Characters association to key pressed/released in default or alphanumeric mode

Below a sequence chart for example :

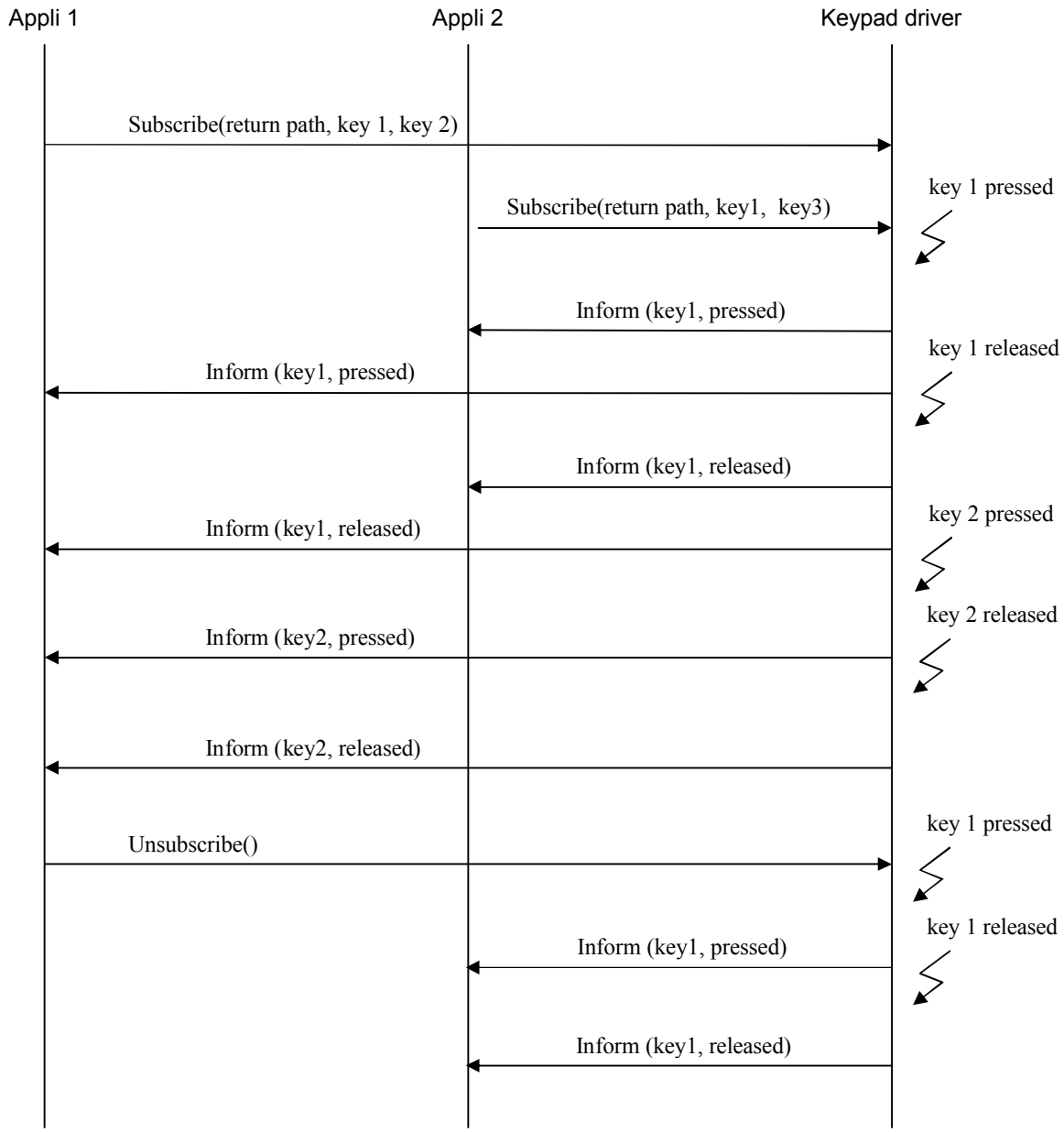


Figure 13: General sequence chart

All the services provided by the keypad SW entity are accessed via direct function call. These functions are listed in this document. The keypad SW entity use the return mechanism defined in the Riviera Environment to provide information back to the client.

Return Mechanism

All the functions return an immediate value, providing information on the success or the failure of the function call. In some (most of the) cases, extra processing time is needed to perform the action

requested when calling the function. In this case, the function is exit and later on, one or several MESSAGEs are sent back by the keypad SW entity.

The keypad SW entity use the MESSAGE format and the return path method defined in Riviera Environment. Basically, in order to send information back, the keypad SW entity sends MESSAGEs to the client. A MESSAGE is a buffer, with a header, common to any MESSAGE, and a custom field related to the MESSAGE. The header is a C structure, containing the *msg_id* field. This field contains the unique msg_id of the MESSAGE and is the only way to know which kind of MESSAGE has been received. Based on this value, the client can re-cast the buffer and access to custom information related to the MESSAGE.

Client have two ways to get access to the MESSAGEs:
Call back functions or message posted with its ADDRESS ID.

A call back function is a function name, provided by client as a parameter and which will be called by the keypad SW when an MESSAGE occurs. When a callback function is defined, it is always the callback function mechanism that is used to return MESSAGE to the client.
But for more efficient implementation, it also possible to directly send a message to the client. In this case, the ADDR ID of the client must be provided to the keypad SW entity. That implies that the client is a Riviera SW entity.

The client can define which return mechanism should be used. For that purpose, it must provide a *return_path*. The generic *return_path* type is a C structure, defined as:

```
typedef struct {
                T_RVF_ADDR_ID                                   addr_id;
                VOID                                                    (*callback
                }T_RV_RETURN;
```

Software design

Constants

Label	Type	Definition
KPD_NB_PHYSICAL_KEYS	Integer (>0)	Define the number of physical keys on the keypad

Note that this constant is in kpd_cfg.h file for keypad driver configuration.

Types definition

T_KPD_SUBSCRIBER

This type defines a subscriber identification for using keypad driver services.
This identification is set by kpd_subscribe function.

typedef void* T_KPD_SUBSCRIBER

T_KPD_VIRTUAL_KEY_ID

This type defines a virtual key identification.

typedef UINT8 T_KPD_VIRTUAL_KEY_ID

T_KPD_VIRTUAL_KEY_TABLE

This structure defines a set of keys available in a particular mode.

```
typedef struct {    UINT8 nb_notified_keys;
                    T_KPD_VIRTUAL_KEY_ID notified_keys[KPD_NB_PHYSICAL_KEYS];
                    } T_KPD_VIRTUAL_KEY_TABLE
```

T_KPD_MODE

This type list all the mode (default, alphanumeric, and customer mode) possibly used by the client.

Available values are:
- KPD_DEFAULT_MODE
- KPD_ALPHANUMERIC_MODE
- <Other customer mode>

T_KPD_NOTIF_LEVEL

This type list different kind of available notification for a key event (cf § **Error! Reference source not found.**).

Available values are:
- KPD_NO_NOTIF
- KPD_FIRST_PRESS_NOTIF
- KPD_LONG_PRESS_NOTIF
- KPD_INFINITE_REPEAT_NOTIF
- KPD_RELEASE_NOTIF

T_KPD_KEY_STATE

This type defines the state pressed and released.

Available values are:
- KPD_KEY_PRESSED
- KPD_KEY_RELEASED

T_KPD_PRESS_STATE

This type defines the particular state for a pressed key.

Available values are:
- KPD_FIRST_PRESS
- KPD_LONG_PRESS
- KPD_REPEAT_PRESS
- KPD_INSIGNIFICANT_VALUE (used when key is released)

T_KPD_KEY_INFO

This type give key information when its state change (pressed ↔ release)
typedef struct { T_KPD_VIRTUAL_KEY_ID virtual_key_id

T_KPD_KEY_STATE state;

T_KPD_PRESS_STATE press_state;
char* ascii_value_p
} T_KPD_KEY_INFO

Messages definition

T_KPD_KEY_EVENT_MSG

This message is sent to a client when a key is pressed or released.

```
typedef struct {
              T_RV_HDR            hdr;
              T_KPD_KEY_INFO      key_info;
              }T_KPD_KEY_EVENT_MSG;
```

T_KPD_STATUS_MSG

This message is sent to a client to return the status of an asynchronous process requested by a client.

```
typedef struct {
              T_RV_HDR     hdr;
              UINT8        operation;
              UINT8        status_value;
              }T_KPD_KEY_EVENT_MSG;
```

Interface functions description

kpd_subscribe

```
T_RV_RET kpd_subscribe (T_KPD_SUBSCRIBER* subscriber_p,
                        T_KPD_MODE mode,
                        T_KPD_VIRTUAL_KEY_TABLE* notified_keys_p,
                        T_RV_RETURN return_path)
```

Description

This function needs to be called by the client before any use of the keypad driver services.
If number of notified key is KPD_NB_PHYSICAL_KEYS, client has not to fulfil the structure, this will be automatically done by the SWE.
By default, client is notified only when the key is released.

Parameters

- **T_KPD_SUBSCRIBER**

Subscriber identification value set by keypad driver to use all the services.

- **T_KPD_MODE**

Mode used by the keypad client (default, alphanumeric, or one amongst those defined by customer).

- **T_KPD_VIRTUAL_KEY_TABLE**

Define all the keys the client want to be notified.

- **T_RV_RETURN**

Return path for key pressed or key released notification.
C.f. paragraph 5.2 : Return Mechanism.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_INTERNAL_ERR	- Max of subscriber is reached, - Software entity is not started, not yet initialized or initialization has failed
RV_INVALID_PARAMETER	Number of virtual keys is not correct
RV_MEMORY_ERR	Memory reaches its size limit

Event Return

- **KPD_STATUS_MSG**

This event is the status sent at the end of the subscription or if an error occurred.
C.f. paragraph 0 for structure definition

The value of *operation* is : KPD_SUBSCRIBE_OP

The possible values of *status_value* are:

id	Definition
KPD_PROCESS_OK	The asynchronous operation is successful
KPD_ERR_KEYS_TABLE	At least one key is not available in the requested mode

KPD_ERR_RETURN_PATH_EXISTING	Subscriber return path is already defined by another subscriber
KPD_ERR_INTERNAL	An internal error occured

- **KPD_KEY_EVENT_MSG**

This event is sent each time a key changes its state or when repetition key event occurs.
C.f. paragraph 0 for structure definition

The value of *virtual_key_id* is the virtual key id.

The possible values of *state* are:

id	Definition
KPD_KEY_PRESSED	Key is pressed
KPD_KEY_RELEASED	Key is released

The possible values of *press_state* are:

id	Definition
KPD_FIRST_PRESS	First press of the key
KPD_LONG_PRESS	Long press of the key
KPD_REPEAT_PRESS	Repeat press of the key
KPD_INSIGNIFICANT_VALUE	Defined when key is released

The value of *ascii_value_p* is the ASCII code associated to the virtual key

Current restriction of use

None.

kpd_unsubscribe

```
T_RV_RET kpd_unsubscribe ( T_KPD_SUBSCRIBER* subscriber_p)
```

Description

This function unsubscribes a client from the keypad driver.

Parameters

- **T_KPD_SUBSCRIBER**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	Subscriber identification is incorrect
RV_MEMORY_ERR	Memory reaches its size limit

Event Return

No message is returned for asynchronous process.

Current restriction of use

None.

kpd_define_key_notification

```
T_RV_RET
kpd_define_key_notification (   T_KPD_SUBSCRIBER subscriber,
                                T_KPD_VIRTUAL_KEY_TABLE* notif_key_table_p,
                                T_KPD_NOTIF_LEVEL notif_level,
                                UINT16 long_press_time,
                                UINT16 repeat_time)
```

Description

This function defines notification level for a set of keys. By default, at subscription, all the keys are defined as KPD_RELEASE_NOTIF for notification level.
It's not mandatory that all the keys defined in the notif_key_table are notified to the subscriber. If one or more key is set in this table but is not notified to the subscriber, this will have no effect.
If number of notified key is KPD_NB_PHYSICAL_KEYS, client has not to fulfil the structure, this will be automatically done by the KPD.

Parameters

- **T_KPD_SUBSCRIBER**

Subscriber identification value.

- **T_KPD_VIRTUAL_KEY_TABLE**

Set of keys for level notification definition.

- **T_KPD_NOTIF_LEVEL**

Define what kind of notification is set for all the keys. Mix of the following values can be used. There are five different values:

- KPD_NO_NOTIF : The client is not notified by any event for this key.
- KPD_FIRST_PRESS_NOTIF The client is not notified of.
  - the immediate key press,
- KPD_LONG_PRESS_NOTIF : The client is notified of
  - the long press if the key is still pressed after "long_press_time",
- KPD_INFINITE_REPEAT_NOTIF : The client is notified of
  - the long press if the key is still pressed after "long_press_time",
  - the key pressed every "repeat_time" tenth of seconds, until key is released,
- KPD_RELEASE_NOTIF : The client is notified of
  - the key release.

- **UINT16 (long_press_time)**

Time tenth of seconds before long press time notification.

- **UINT16 (repeat_time)**

Time in tenth of seconds for key repetition.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	- Subscriber identification is incorrect - Number of virtual keys is not correct - long_press_time = 0 and repeat_level = KPD_LONG_PRESS_NOTIF or KPD_INFINITE_REPEAT_NOTIF - repeat_time=0 and repeat_level = KPD_INFINITE_REPEAT_NOTIF
RV_MEMORY_ERR	Memory reaches its size limit

Event Return

- **KPD_STATUS_MSG**

This event is the status sent at the end of the repeat key definition or if an error occurred.
C.f. paragraph 0 for structure definition

The value of *operation* is : KPD_REPEAT_KEYS_OP

The possible values of *status_value* are:

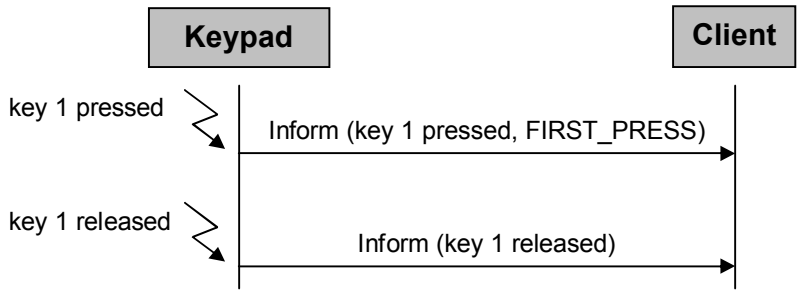
id	Definition
KPD_PROCESS_OK	The asynchronous operation is successful
KPD_ERR_KEYS_TABLE	At least one key is not available in the subscriber mode

Current restriction of use

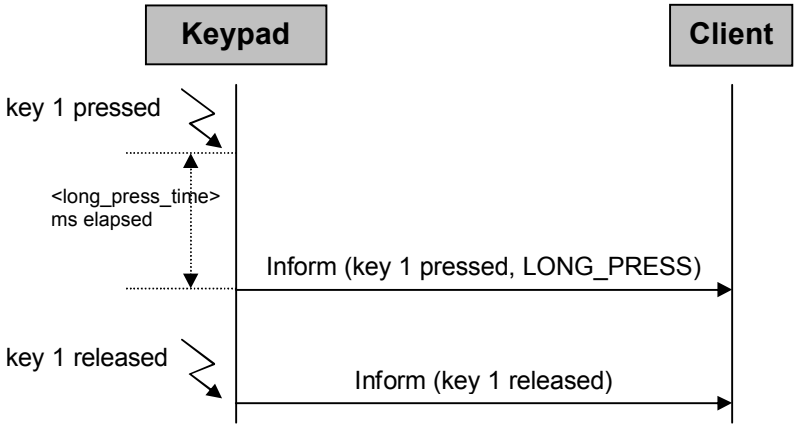
Values for long_press_time and repeat_time are available for the subscriber but for all the key defined in repeat mode. So if a subscriber call the function twice with different values for long_press_time and repeat_time, only latest values will be taken into account.

Process flow

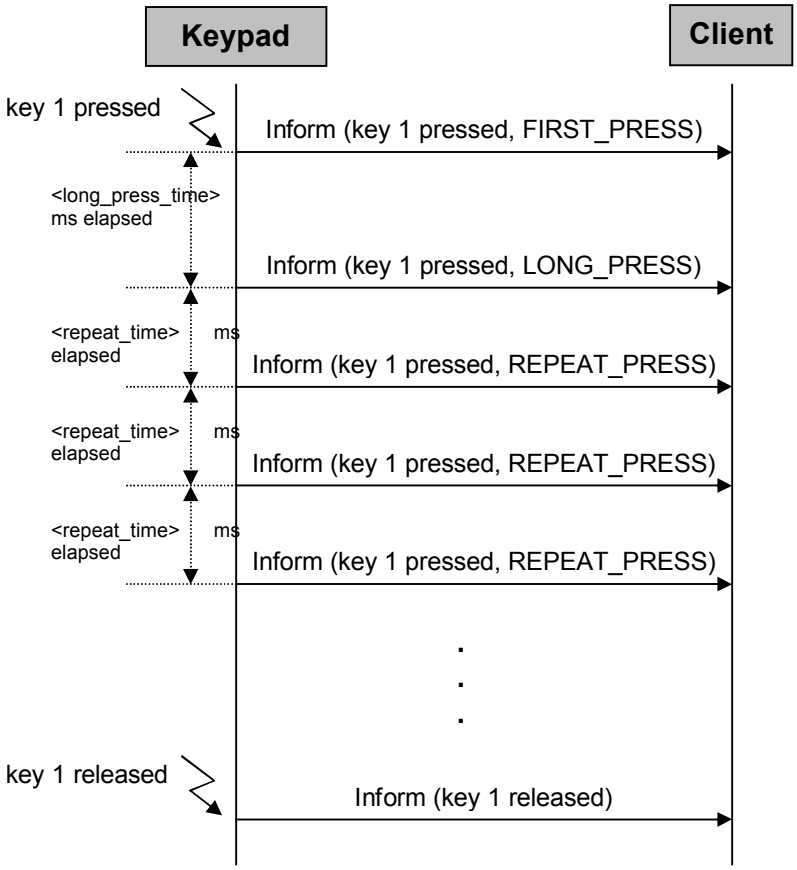
- Key defined for “first press” and “release”



- Key defined for “long press” and “release”



- Key defined for “infinite repeat”, “first press” and “release”



kpd_change_mode

```
T_RV_RET kpd_change_mode(   T_KPD_SUBSCRIBER subscriber,
                            T_KPD_VIRTUAL_KEY_TABLE* notified_keys_p,
                            T_KPD_MODE new_mode)
```

Description

This function changes the mode for the specific client.
If number of notified key is KPD_NB_PHYSICAL_KEYS, client has not to fulfil the structure, this will be automatically done by the KPD.

Parameters

- **T_KPD_SUBSCRIBER**

Subscriber identification value.

- **T_KPD_VIRTUAL_KEY_TABLE**

Define all the keys the client want to be notified in the new mode.

- **T_KPD_MODE**

New mode in which the client want to switch.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	- Subscriber identification value is incorrect, - Number of virtual keys is not correct
RV_MEMORY_ERR	Memory reaches its size limit

Event Return

- **KPD_STATUS_MSG**

This event is the status sent at the end of the change mode request or if an error occurred.
C.f. paragraph 0 for structure definition

The value of *operation* is : KPD_CHANGE_MODE_OP

The possible values of *status_value* are:

id	Definition
KPD_PROCESS_OK	The asynchronous operation is successful
KPD_ERR_KEYS_TABLE	At least one key is not available in the new requested mode

Current restriction of use

Call to this function cancel, for the subscriber, all the repeat mode defined for the keys with the function kpd_define_repeat_key.

If the subscriber was the owner of the keypad, this privilege is cancelled and keypad is set in multi-notified mode.

If RV_INVALID_PARAMETER is returned, the current mode for the subscriber (if it exists) is the old mode.

kpd_own_keypad

```
T_RV_RET kpd_own_keypad ( T_KPD_SUBSCRIBER subscriber,
                          BOOL is_keypad_owner,
                          T_KPD_VIRTUAL_KEY_TABLE* keys_owner_p)
```

Description

This function allows a subscriber being the only client to be notified by action on keypad (less CPU time used).
After this call, the keypad is in the "single notified" state.
This action is cancelled when:

- The function is called with parameter *is_keypad_owner* to FALSE,
- The subscriber (keypad owner) unsubscribes from keypad.
- The subscriber (keypad owner) changes its mode.

Note that keypad is in the "multi notified" state if there is no subscriber (particularly at the keypad initialisation)

Parameters

- **T_KPD_SUBSCRIBER**

Subscriber identification value.

- **BOOL**

Define the state to change

- TRUE : keypad pass in "single notified" state
- FALSE : keypad pass in "multi notified" state

- **T_KPD_VIRTUAL_KEY_TABLE**

Set of keys only notified to the subscriber that calls this function. It is mandatory that this table is a subset of keys defined at the subscription.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	- Subscriber identification value is incorrect, - Number of virtual keys is not correct
RV_MEMORY_ERR	Memory reaches its size limit

Event Return

- **KPD_STATUS_MSG**

This event is the status sent at the end of the own keypad request or if an error occurred.
C.f. paragraph 0 for structure definition

The value of *operation* is : KPD_OWN_KEYPAD_OP

The possible values of *status_value* are:

id	Definition
KPD_PROCESS_OK	The asynchronous operation is successful
KPD_ERR_KEYS_TABLE	At least one key is not defined in the subscriber mode

KPD_ERR_SN_MODE	Keypad driver is already in SN mode
KPD_ERR_ID_OWNER_KEYPAD	Subscriber try to remove own keypad privilege but it is not the keypad owner.

Current restriction of use

None

kpd_set_key_config

```
T_RV_RET kpd_set_key_config (  T_KPD_SUBSCRIBER          subscriber,
                               T_KPD_VIRTUAL_KEY_TABLE* reference_keys_p,
                               T_KPD_VIRTUAL_KEY_TABLE* new_keys_p)
```

Description

This function allows setting dynamically a configuration for new or existing virtual keys. The two tables define a mapping between each entry (new_keys[1] is mapped with reference_keys[1], new_keys[2] is mapped with reference_keys[2], …).
The call of this function doesn't change the mode of the client.

Parameters

- **T_KPD_SUBSCRIBER**

Subscriber identification value.

- **T_KPD_VIRTUAL_KEY_TABLE (reference_keys_p)**

Set of keys available on keypad in default mode.

- **T_KPD_VIRTUAL_KEY_TABLE (new_keys_p)**

Set of keys that must map with the reference keys.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	- Subscriber identification value is incorrect, - At least, one reference key is not defined in the default mode - Number of virtual keys is not correct (in reference key or new key table)
RV_NOT_SUPPORTED	Configurable mode is not supported.
RV_MEMORY_ERR	Memory reaches its size limit

Event Return

- **KPD_STATUS_MSG**

This event is the status sent at the end of the ey configuration request or if an error occurred.
C.f. paragraph 0 for structure definition

The value of *operation* is : KPD_SET_CONFIG_MODE_OP

The possible values of *status_value* are:

id	Definition
KPD_PROCESS_OK	The asynchronous operation is successful
KPD_ERR_CONFIG_MODE_USED	Some subscribers use Config mode.

Current restriction of use

None

kpd_get_available_keys

```
T_RV_RET kpd_get_available_keys (  T_KPD_VIRTUAL_KEY_TABLE*
available_keys_p)
```

Description

This function allows knowing all the available keys in default mode.

Parameters

- **T_KPD_VIRTUAL_KEY_TABLE**

Set of keys available on keypad in default mode. The structure must be declared by the caller, and is filled by the function.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.

Event Return

None.

Current restriction of use

None.

kpd_get_ascii_key_code

```
T_RV_RET kpd_get_ascii_key_code (   T_KPD_VIRTUAL_KEY_ID key,
                                    T_KPD_MODE mode,
                                    char** ascii_code_pp)
```

Description

This function return associated ASCII value to defined key.

Parameters

- **T_KPD_VIRTUAL_KEY_ID**

Define key identification value.

- **T_KPD_MODE**

Mode in which is defined the link between “key” and “ascii_code”

- **char (ascii_code_pp)**

Associated ASCII code to parameter “key”.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	- Mode is different of KPD_DEFAULT_MODE or KPD_ALPHANUMERIC_MODE - Key doesn’t exist in the defined mode

If returned value is RV_INVALID_PARAMETER, empty string is set in ascii_code_pp variable.

Event Return

None.

Current restriction of use

None.

KP_Init (Deprecated function)

```
void KP_Init ( void(pressed(T_KPD_VIRTUAL_KEY_ID)),
               void(released(void)) )
```

Description

This function is defined for backward compatibility with Condat.
It register two functions which notify Condat that Power key is pressed for power ON, power OFF indication.
For information, virtual key id of power key is **KPD_PWR**

Parameters

- **pressed**

Function called when power key is pressed.

- **released**

Function called when power key is released.

Immediate Return

None

Event Return

None.

Current restriction of use

This function will be removed as soon as possible, when Condat will define one or two function to call when mobile is power ON and mobile is power OFF.

kpd_retrieve_key_status

```
T_RV_RET kpd_retrieve_key_status(  T_KPD_VIRTUAL_KEY_ID key_id,
                                                T_KPD_MODE mode,
                                                T_KPD_KEY_STATE* state);)
```

Description

This function allows application to check the status of a key (whether pressed or released).This API has been newly added to fix KPD release interrupt miss issue.

Parameters

- **key_id**

Function called when power key is pressed.

- **mode**

Function called when power key is released.

- **state**

This will habe the status of the key (KPD_KEY_RELEASED or KPD_KEY_PRESSED).

Immediate Return

RV_OK if operation is successfull,
RV_INVALID_PARAMETER if :
- mode is different of KPD_DEFAULT_MODE or KPD_ALPHANUMERIC_MODE,
- the key doesn't exist in the defined mode.

Event Return

None.

Chapter 18 RTC

18.1 Introduction

This chapter provides an interface specification of the RTC SW entity.

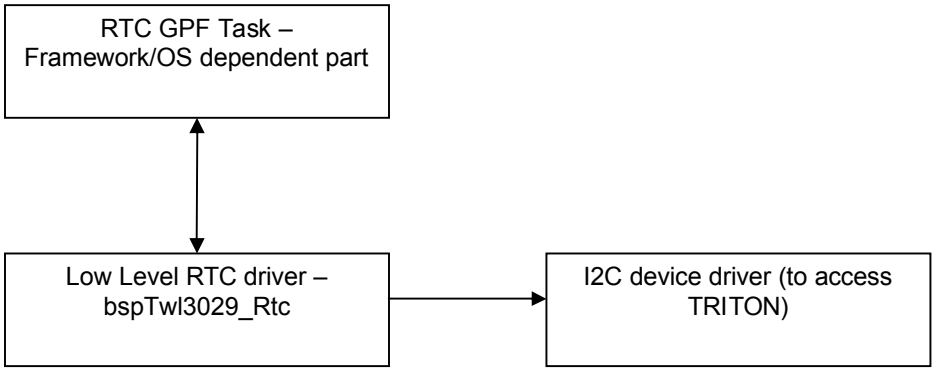
18.1.1 Generality

The main purpose of the RTC driver is to provide the current date and time related functions incorporating features like setting and unsetting of alarm .

RTC device driver is divided into two parts,

1. Framework and OS agnostic part and
2. Framework/OS dependent part.

All access to the RTC device on the TRITON happens through the I2C driver.

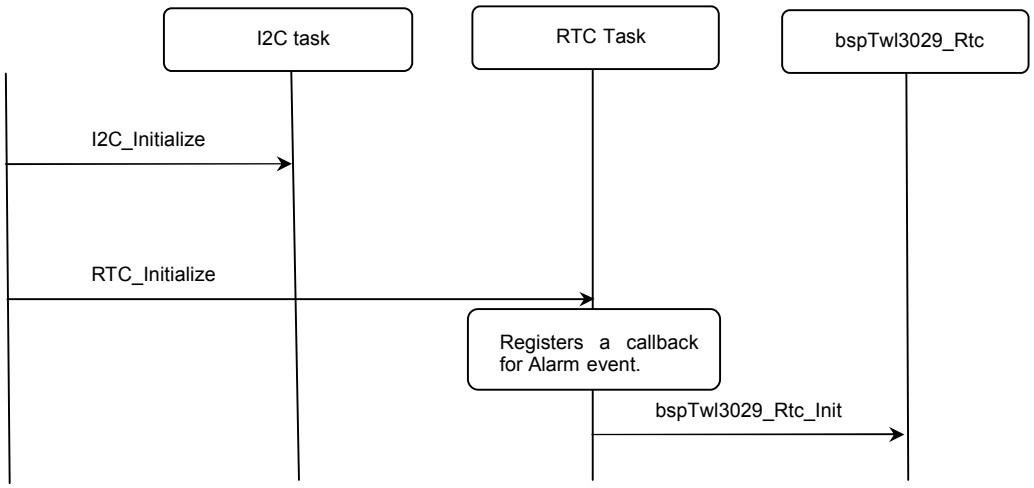


All the APIs defined are framework agnostic. All APIs that use framework related functions are given a wrapper.The wrapper function takes care of framework related functions. In other case where,

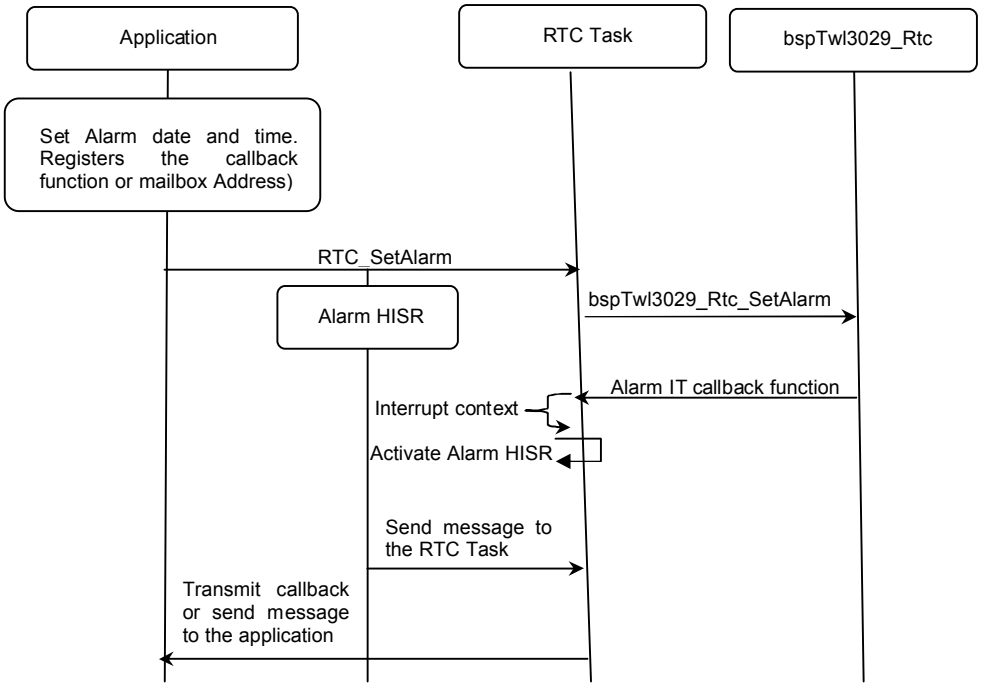
1. HISR is activated from ISR
2. Message is sent from ISR

A call-back function is registered to this Framework agnostic driver which is called from ISR context. This call-back function handles the HISR activation or message sending.

18.1.2 Sequence Diagrams



RTC initialization



Set Alarm Sequence diagram

18.2 Interface description

18.2.1 RTC_INITIALIZE

```
T_RVF_RET RTC_INITIALIZE(void)
```

Description

This function does initialization and starts the RTC by calling bspTwl3029_Rtc_Init. This registers a call-back to the bspTwl3029_Rtc driver which is called when an interrupt occurs.

Parameters

None

Immediate Return

- **T_RVF_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_INTERNAL_ERR	-Triton Initialization function failed.
RV_MEMORY_ERR	Memory allocation for alarm event failed

Event Return

None

Current restriction of use

This is the first API to be called before the useof any API of RTC.

18.2.2 RTC_RtcReset

```
BOOL RTC_RtcReset(void)
```

Description

This function indicates if RTC reset has occurred. This function calls the bspTwl3029_Rtc_IsReset.

Parameters

None

Immediate Return

- **BOOL**

The possible values are:

id	Definition
TRUE	RESET has successfully executed.
FALSE	RESET Failed

Event Return

None

Current restriction of use

None

18.2.3 RTC_GetDateTime

```
T_RVF_RET RTC_GetDateTime (T_RTC_DATE_TIME* date_time)
```

Description

This function provides the current date and time by calling the bspTwl3029_Rtc_GetDateTime .

Parameters

- **T_RTC_DATE_TIME**

This parameter, a structure, is updated by the API to provide date and time.

```
typedef struct {  UINT8   second;      /* seconds after the minute   - [0,59]  */
                  UINT8   minute;      /* minutes after the hour     - [0,59]  */
                  UINT8   hour;        /* hours after the midnight   - [0,23]  */
                  UINT8   day;         /* day of the month           - [1,31]  */
                  UINT8   month;       /* months                     - [01,12] */
                  UINT8   year;        /* years                      - [00,99] */
                  UINT8   wday;        /* days in a week             - [0,6] */
                  BOOL    mode_12_hour; /* TRUE->12 hour mode ; FALSE-> 24 hour mode */
                  BOOL    PM_flag;     /* if 12 hour flag = TRUE
                 TRUE->PM ; FALSE->AM */
               } T_RTC_DATE_TIME;
```

Immediate Return

- **T_RVF_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RVF_INTERNAL_ERR	bspTwl3029_Rtc_GetDateTime function failed.

Event Return

None

Current restriction of use

None

18.2.4 RTC_SetDateTime

```
T_RVF_RET RTC_SetDateTime(T_RTC_DATE_TIME date_time)
```

Description

This function sets date and time into RTC registers using bspTwl3029_Rtc_SetDateTime.

Parameters

- **T_RTC_DATE_TIME**

This parameter, a structure, is updated by the API to provide date and time.

```
typedef struct {  UINT8   second;      /* seconds after the minute   - [0,59]  */
                  UINT8   minute;      /* minutes after the hour     - [0,59]  */
                  UINT8   hour;        /* hours after the midnight   - [0,23]  */
                  UINT8   day;         /* day of the month           - [1,31]  */
                  UINT8   month;       /* months                     - [01,12] */
                  UINT8   year;        /* years                      - [00,99] */
                  UINT8   wday;        /* days in a week             - [0,6]   */
                  BOOL    mode_12_hour; /* TRUE->12 hour mode ; FALSE-> 24 hour mode */
                  BOOL    PM_flag;     /* if 12 hour flag = TRUE
                                          TRUE->PM ; FALSE->AM */
               } T_RTC_DATE_TIME;
```

Immediate Return

- **T_RVF_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_INTERNAL_ERR	-Triton bspTwl3029_Rtc_SetDateTime function failed.

Event Return

None

Current restriction of use

None

18.2.5 RTC_GetAlarm

```
T_RVF_RET RTC_GetAlarm(T_RTC_DATE_TIME* date_time)
```

Description

This function provides the current alarm value.

Parameters

- **T_RTC_DATE_TIME**

This parameter, a structure, is updated by the API to provide date and time.

```
typedef struct {  UINT8   second;      /* seconds after the minute   - [0,59]  */
                  UINT8   minute;      /* minutes after the hour     - [0,59]  */
                  UINT8   hour;        /* hours after the midnight   - [0,23]  */
                  UINT8   day;         /* day of the month           - [1,31]  */
                  UINT8   month;       /* months                     - [01,12] */
                  UINT8   year;        /* years                      - [00,99] */
                  UINT8   wday;        /* days in a week             - [0,6] */
                  BOOL    mode_12_hour; /* TRUE->12 hour mode ; FALSE-> 24 hour mode */
                  BOOL    PM_flag;     /* if 12 hour flag = TRUE
                                          TRUE->PM ; FALSE->AM */
               } T_RTC_DATE_TIME;
```

Immediate Return

- **T_RVF_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.

Event Return

None

Current restriction of use

None.

18.2.6 RTC_SetAlarm

```
T_RVF_RET RTC_SetAlarm(T_RTC_DATE_TIME date_time, T_RV_RETURN return_path)
```

Description

This sets alarm date and time using bspTwl3029_Rtc_SetAlarm function.

Parameters

- **T_RV_RETURN**

Return path for Alarm event notification

- **T_RTC_DATE_TIME**

This parameter, a structure, is updated by the API to provide date and time.

```
typedef struct {  UINT8   second;      /* seconds after the minute   - [0,59]  */
                  UINT8   minute;      /* minutes after the hour     - [0,59]  */
                  UINT8   hour;        /* hours after the midnight   - [0,23]  */
                  UINT8   day;         /* day of the month           - [1,31]  */
                  UINT8   month;       /* months                     - [01,12] */
                  UINT8   year;        /* years                      - [00,99] */
                  UINT8   wday;        /* days in a week             - [0,6] */
                  BOOL    mode_12_hour; /* TRUE->12 hour mode ; FALSE-> 24 hour mode */
                  BOOL    PM_flag;     /* if 12 hour flag = TRUE
                                          TRUE->PM ; FALSE->AM */
               } T_RTC_DATE_TIME;
```

Immediate Return

- **T_RVF_RET**

The possible values are:

Id	Definition
RVF_OK	The API function successfully executed.
RVF_INTERNAL_ERR	-Triton bspTwl3029_Rtc_SetAlarm function failed.

Event Return

None

Current restriction of use

None

18.2.7 RTC_UnsetAlarm

```
T_RVF_RET RTC_UnsetAlarm(void)
```

Description

This disables the alarm interrupts, thus preventing the alarm event from occurring, using bspTwl2039_Rtc_UnsetAlarm.

Parameters

None

Immediate Return

- **T_RVF_RET**

The possible values are:

Id	Definition
RVF_OK	The API function successfully executed.
RVF_INTERNAL_ERR	-Triton bspTwl3029_Rtc_UnsetAlarm function failed.

Event Return

None

Current restriction of use

None

18.2.8 RTC_Rounding30s

```
void RTC_Rounding30s(void)
```

Description

This API calls the bspTwl3029_Rtc_Rounding30s to rounds the current time to the nearest minute.

Parameters

None

Immediate Return

None

Event Return

None

Current restriction of use

None

18.2.9 RTC_Set12HourMode

```
void RTC_Set12HourMode(BOOL Mode12Hour)
```

Description

This API calls bspTwl3029_Rtc_Set12HourMode to set the hour in 24 hour or 12 hour mode

Parameters

- **BOOL**

Accept whether the desired format should be in 12hour format(TRUE) or in 24hour format(FALSE)

Immediate Return

None

Event Return

None

Current restriction of use

None

18.2.10 RTC_Is12HourMode

```
BOOL RTC_Is12HourMode(void)
```

Description

This calls the bspTwl3029_Rtc_Is12HourMode to get the current hour mode.

Parameters

None

Immediate Return

- **BOOL**

This returns whether the RTC is in 12hour mode(TRUE) or in 24hour mode(FALSE).

Event Return

None

Current restriction of use

None

Chapter 19 MPK

19.1 Introduction

This chapter provides an interface specification of the MPK .

19.1.1 Generality

The Manufacturer Public Key driver (MPK) provides a service to read the Manufacturer Public Key register. The MPK-identifier is a 128-bit register composed of fuse cells electrically programmed and enclosing the 128 lsb of the hashing value (SHA-1) of the Public Key component of the manufacturer's Public Key pair. This value is used to authenticate any certificate signed with the Private Key from the Manufacturer.

19.2 Interface description

19.2.1 mpk_get_mpk_id

```
T_RV_RET mpk_get_mpk_id (UINT8* id_p)
```

Description

This function returns the Manufacturer Public Key id 128-bit value.

Parameters

- **id_p**

Pointer to client buffer of at least 16 bytes.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_INTERNAL_ERR	Pointer to client buffer is NULL

Event Return

None

Current restriction of use

None.

19.2.2 mpk_get_sw_version

```
UINT32 mpk_get_sw_version(void)
```

Description

This function returns the software version of the driver.

Parameters

None

Immediate Return

- **UINT32**

Software Version number is returned. The version is hard coded in BCD format within the software.

Event Return

None

Current restriction of use

None.

Chapter 20 GBI

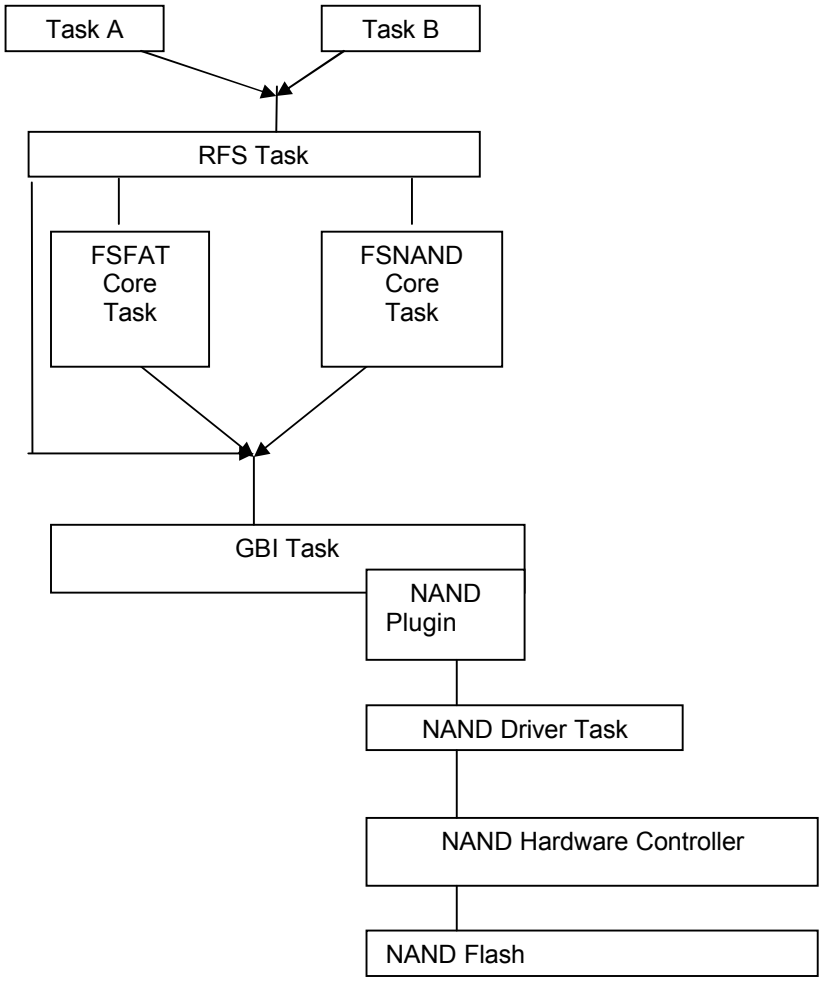
20.1 Introduction

This document provides an interface specification of the GBI.

20.1.1 Generality

The GBI is a generic interface on top of storage devices drivers and provides a block oriented data exchange. The functionality is required for file management in the Universal File System (UFS) concept of Texas Instruments Riviera environment. Other parts of the UFS is the Riviera File System (RFS) and the different File system Cores.

The GBI Task interacts with media drivers such as NAND and MMC/SD.
A single media driver can be able to handle more then one media devices (like several MMC cards). It is also possible that a single media device can have more then one partition. It is also possible that a single media device can have more then one partition.



20.1.2 Detailed Design of GBI API functions

The GBI API provides bridge functions and some non-bridge functions to the client.

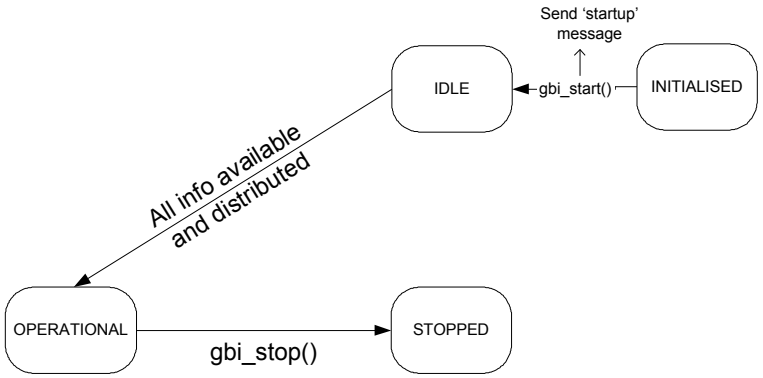
Bridge functions

Bridge functions transfer control of the operation to the GBI entity by sending the function request as an internal message. Execution of the functionality is than done in the context of this entity. In this way the client is not blocked and the execution is done asynchronous. When the GBI entity is ready, it notifies this to the client in the way that he has requested it (call-back or message).

Non-bridge functions
Some functions of the GBI API are non-bridge functions or can be used as non-bridge functions. If a requested functionality can be fulfilled without significant delay, it can be done in the client context. Because of the fast execution it is then non-blocking. If the execution takes considerable time and the non-bridge function is not used asynchronous, the function will be blocking. The execution in the client context is then postponed (sleep/wait) and in the GBI context continued (like bridge functions by internal messages). When GBI is ready, the client task becomes ready for execution again.

Reantrance
All GBI API functions are reentrance. Whenever atomic data operations or execution sequences are in risk of interruption, a mutex is used to guarantee that it stays atomic.

20.1.3 Design of GBI API functions



20.2 Interface description

20.2.1 gbi_read

```
T_RV_RET gbi_read (UINT8           media_nmb,
                   UINT8           partition_nmb,
                   T_GBI_BLOCK     first_block_nmb,
                   T_GBI_BLOCK     number_of_blocks,
                   T_GBI_BYTE_CNT  remainder_length,
                   UINT32          *buffer_p,
                   T_RV_RETURN     return_path)
```

Description
This function reads a number of data blocks from the partition on the specified media. The data is copied to the buffer that is to be reserved by the client. The read starts from the first logical block first_block_nmb and reads number_of_blocks blocks. After the last block it completes by reading remainder_length bytes.

Parameters

- **media_nmb**

This identifies the media for which the command is intended. This number is provided when retrieving the overall partition table from the GBI.

- **partition_nmb**

This identifies the partition for which the command is intended. The partition must be located on the media specified by the media_nmb parameter. This number is provided when retrieving the overall partition table from the GBI .

- **first_block_nmb**

The first logical block number from where the data is requested.

- **number_of_blocks**

The number of logical blocks to read. The number may be from one to the last possible block number. The block size (the number of bytes per block) can be obtained by reading overall partition table from the GBI.

- **remainder_length**

The number of bytes to read after the last whole block. A value of zero indicates no remainder.

- **buffer_p**

This is a pointer to the buffer to which the data shall be copied.

- **return_path**

This structure provides information about the way the driver must react asynchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_MEMORY_ERR	Pointer to client buffer is NULL
RV_INVALID_PARAMETER	Both Callback function and address id are absent .
RVF_INTERNAL_ERR	GSPTaskIdTable[id] returns value zero.

Event Return

- **T_GBI_READ_RSP_MSG**

```
typedef struct {  T_RV_HDR          hdr;
                  T_RV_RET          result;
                 } T_GBI_READ_RSP_MSG;
```

Possible values of result are :

id	Definition
RV_OK	The asynchronous operation is successful
RV_INVALID_PARAMETER	One ore more of the parameters is incorrect (invalid media_nmb, partition_nmb, first_block_nmb out of range, number_of_blocks out of range, remainder_length out of range, buffer_p is null pointer).

RV_NOT_READY	The driver is not able to handle this request at this moment (TASK not initialised).
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check if the media is removed or exchanged.

Current restriction of use

None.

20.2.2 gbi_write

```
T_RV_RET gbi_write (UINT8           media_nmb,
                    UINT8           partition_nmb,
                    T_GBI_BLOCK     first_block_nmb,
                    T_GBI_BLOCK     number_of_blocks,
                    T_GBI_BYTE_CNT  remainder_length,
                    UINT32          *buffer_p,
                    T_RV_RETURN     return_path)
```

Description

This function writes a number of data blocks to the partition on the specified media. The data is copied to the buffer, which is to be reserved by the client. The write starts from the first logical block first_block_nmb and continues for number_of_blocks blocks. After this, remainder_length bytes are written.The function returns immediately. Only the return path is verified immediately.Processing is done asynchronous and the result is returned through the return path.If the media requires this, the driver shall erase the blocks prior to the actual write.

Parameters

- **media_nmb**

This identifies the media for which the command is intended. This number is provided when retrieving the overall partition table from the GBI.

- **partition_nmb**

This identifies the partition for which the command is intended. The partition must be located on the media specified by the media_nmb parameter. This number is provided when retrieving the overall partition table from the GBI .

- **first_block_nmb**

The first logical block number to where the data must be written.

- **number_of_blocks**

The number of logical blocks to write. The number may be from one to the last possible block number. The block size (the number of bytes per block) can be obtained by reading overall partition table from the GBI.

- **remainder_length**

The number of bytes to write after the last whole block. A value of zero indicates no remainder.

- **buffer_p**

This is a pointer to the buffer from where the data will be copied.

- **return_path**

This structure provides information about the way the driver must react asynchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_MEMORY_ERR	Allocation for messge buffer failed.
RV_INVALID_PARAMETER	Both Callback function and address id are absent .
RVF_INTERNAL_ERR	Gpf_Send_msg failed.

Event Return

- **T_GBI_WRITE_RSP_MSG**

```
typedef struct {  T_RV_HDR      hdr;
                  T_RV_RET      result;
                 } T_GBI_WRITE_RSP_MSG;
```

Possible values of result are :

id	Definition
RV_OK	Success
RV_INVALID_PARAMETER	One ore more of the parameters is incorrect. (Invalid media_nmb, partition_nmb, *first_block_nmb* out of range, *number_of_blocks* out of range, *remainder_length* out of range, *buffer_p* is null pointer).
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_NOT_READY	The driver is not able to handle this request at this moment (SWE not initialised).
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check if the media is removed or exchanged.

Current restriction of use

None.

20.2.3 gbi_write_with_spare

```
T_RV_RET gbi_write_with_spare (UINT8           media_nmb,
                               UINT8           partition_nmb,
                               T_GBI_BLOCK     first_block_nmb,
                               T_GBI_BLOCK     number_of_blocks,
                               T_GBI_BYTE_CNT remainder_length,
                               UINT32          *data_buffer_p,
                               UINT32          *spare_buffer_p,
                               T_RV_RETURN     return_path)
```

Description

This function writes a number of data blocks to the partition on the specified media. The data is copied to the buffer, which is to be reserved by the client. The write starts from the first logical block first_block_nmb and continues for number_of_blocks blocks. After this, remainder_length bytes are written.When spare_buffer_p is used for each block the spare data will be written. This function is only available for media where spare data is available.The function returns immediately. Only the return path is verified immediately. Processing is done asynchronous and the result is returned through the return path. If the media requires this, the driver shall erase the blocks prior to the actual write.

Parameters

- **media_nmb**

This identifies the media for which the command is intended. This number is provided when retrieving the overall partition table from the GBI.

- **partition_nmb**

This identifies the partition for which the command is intended. The partition must be located on the media specified by the media_nmb parameter. This number is provided when retrieving the overall partition table from the GBI .

- **first_block_nmb**

The first logical block number to where the data must be written.

- **number_of_blocks**

The number of logical blocks to write. The number may be from one to the last possible block number. The block size (the number of bytes per block) can be obtained by reading overall partition table from the GBI.

- **remainder_length**

The number of bytes to write after the last whole block. A value of zero indicates no remainder.

- **buffer_p**

This is a pointer to the buffer from where the data will be copied.

- **spare_buffer_p**

A pointer to the buffer from where the spare data is copied.

- **return_path**

This structure provides information about the way the driver must react asynchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_MEMORY_ERR	Allocation for messge buffer failed.
RV_INVALID_PARAMETER	Both Callback function and address id are absent .
RVF_INTERNAL_ERR	Gpf_Send_msg failed.

Event Return

- **T_GBI_WRITE_WITH_SPARE_RSP_MSG**

```
typedef struct {  T_RV_HDR              hdr;
                  T_RV_RET              result;
                } T_GBI_WRITE_SPARE_DATA_RSP_MSG;
```

Possible values of result are :

id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not able to handle this request at this moment.
RV_INVALID_PARAMETER	One or more parameters are invalid (media_nmb, partition_nmb is incorrect, *first_block* is out of range, *number_of_blocks* is out of range, *info_data_p* pointer is a null pointer, *offset_first_byte* is behind size of the spare area, *nmb_bytes* in combination with *offset_first_byte is* behind size of the spare area)
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request
RV_INTERNAL_ERROR	There was an internal error while executing the request.
RV_NOT_SUPPORTED	The driver does not support the spare data feature.

Current restriction of use

None.

20.2.4 gbi_erase

```
T_RV_RET gbi_erase (UINT8           media_nmb,
                    UINT8           partition_nmb,
                    T_GBI_BLOCK     first_block_nmb,
                    T_GBI_BLOCK     number_of_blocks,
                    T_RV_RETURN     return_path)
```

Description

This function erases a number of data blocks from the partition on the specified media. The erase starts from the first logical block first_block_nmb and continues for number_of_blocks blocks. The function returns immediately. Only the return path is verified immediately.Processing is done asynchronous and the result is returned through the return path.

Parameters

- **media_nmb**

This identifies the media for which the command is intended. This number is provided when retrieving the overall partition table from the GBI.

- **partition_nmb**

This identifies the partition for which the command is intended. The partition must be located on the media specified by the media_nmb parameter. This number is provided when retrieving the overall partition table from the GBI .

- **first_block_nmb**

The first logical block number to where the data must be written.

- **number_of_blocks**

The number of logical blocks to erase. The number may be from one to the last possible block number. The block size (the number of bytes per block) can be obtained by reading overall partition table from the GBI.

- **return_path**

This structure provides information about the way the driver must react asynchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_MEMORY_ERR	Allocation for messge buffer failed.
RV_INVALID_PARAMETER	Both Callback function and address id are absent .
RVF_INTERNAL_ERR	Gpf_Send_msg failed.

Event Return

- **T_GBI_ERASE_RSP_MSG**

```
typedef struct {  T_RV_HDR      hdr;
                  T_RV_RET      result;
                 } T_GBI_ERASE_RSP_MSG;
```

The possible values of the result are :

id	Definition
RV_OK	Success
RV_INVALID_PARAMETER	One ore more of the parameters is incorrect. (Invalid media_nmb, partition_nmb, *first_block_nmb* out of range, *number_of_blocks* out of range,).
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_NOT_READY	The driver is not able to handle this request at this moment (Task not initialised).
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check if the media is removed or exchanged.

Current restriction of use

None.

20.2.5 gbi_flush

```
T_RV_RET gbi_flush (UINT8           media_nmb,
                    UINT8           partition_nmb,
                    T_RV_RETURN     return_path)
```

Description

When a client writes data to a media partition, this data is often not immediately written on the physical media. This may be caused by different buffer sizes that are used by various software and hardware components. Another reason may be task scheduling or hardware delays. To ensure that the data is consistent and really written on the physical media, the flush function can be used.The function returns immediately. Only the return path is verified immediately.Processing is done asynchronous and the result is returned through the return path.

Parameters

- **media_nmb**

This identifies the media for which the command is intended. This number is provided when retrieving the overall partition table from the GBI.

- **partition_nmb**

This identifies the partition for which the command is intended. The partition must be located on the media specified by the media_nmb parameter. This number is provided when retrieving the overall partition table from the GBI .

- **return_path**

This structure provides information about the way the driver must react asynchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_MEMORY_ERR	Allocation for messge buffer failed.
RV_INVALID_PARAMETER	Both Callback function and address id are absent .
RVF_INTERNAL_ERR	Gpf_Send_msg failed.

Event Return

- **T_GBI_FLUSH_RSP_MSG**

```
typedef struct {  T_RV_HDR              hdr;
                  T_RV_RET              result;
                 } T_GBI_FLUSH_RSP_MSG;
```

The possible values of result are :

id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not able to handle this request at this moment.
RV_INVALID_PARAMETER	The *media_nmb or partition_nmb* parameter is incorrect.
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check if the media is removed or exchanged.

Current restriction of use

None.

20.2.6 gbi_get_media_info

```
T_RV_RET gbi_get_media_info(T_RVF_MB_ID   mb_id,
                            T_RV_RETURN   return_path)
```

Description

This function return detailed information about all the media known by the GBI entity.The function returns immediately. Only the return_path parameter is verified immediately. Processing (checking and filling the media information structure) is done asynchronous and the result is returned through the return path.When the processing is started, GBI first allocates the client's memory where the media information shall be copied in. After the information is copied into the allocated memory, the response message is send. This message contains the pointer to this media information structure. The client accesses the information through this pointer.The client is responsible for de-allocating the memory when the data is not longer needed.

Parameters

- **mb_id**

Memory bank ID of the client's memory bank (and where the media information shall be copied to).

- **return_path**

This structure provides information about the way the driver must react asynchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_MEMORY_ERR	Allocation for messge buffer failed.
RV_INVALID_PARAMETER	Either Both Callback function and address id are absent or mb_id is incorrect
RVF_INTERNAL_ERR	Gpf_Send_msg failed.

Event Return

- **T_GBI_MEDIA_INFO_RSP_MSG**

```
typedef struct {  T_RV_HDR              hdr;
                  T_RV_RET              result;
                  UINT8                 nmb_of_media; //see note
                  T_GBI_MEDIA_INFO      *info_p; //see note
                  } T_GBI_MEDIA_INFO_RSP_MSG;
```

Note: the members “nmb_of_media” and “info_p” are only valid if the “result” member has the value RV_OK. The “info_p” member is a pointer to an array of structures of type “T_GBI_MEDIA_INFO”. The array is “nmb_of_media” long.

The possible values of result are :

id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not able to handle this request at this moment.
RV_INVALID_PARAMETER	mb_id parameters is incorrect
RV_INTERNAL_ERROR	There was an internal error while executing the request.

Current restriction of use

None.

20.2.7 gbi_get_partition_info

```
T_RV_RET gbi_get_partition_info(T_RVF_MB_ID   mb_id,
                                T_RV_RETURN    return_path);
```

Description

This function return detailed information about all the partitions that are known by the GBI entity.
The function returns immediately. Only the return_path parameter is verified immediately. Processing (copying the partition information) is done asynchronous and the result is returned through the return path. When the processing is started, GBI first allocates the client's memory where the partition information shall be copied in. After the information is copied into the allocated memory, the response message is send. This message contains the pointer to this partition information. The client then accesses the information through this pointer.The client is responsible for de-allocating the memory when the data is not longer needed.

Parameters

- **mb_id**

Memory bank ID of the client's memory bank (and where the media information shall be copied to).

- **return_path**

This structure provides information about the way the driver must react asynchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_MEMORY_ERR	Allocation for messge buffer failed.
RV_INVALID_PARAMETER	Either Both Callback function and address id are absent or mb_id is incorrect
RVF_INTERNAL_ERR	Gpf_Send_msg failed.

Event Return

- **T_GBI_PARTITION_INFO_RSP_MSG**

```
typedef struct {  T_RV_HDR                    hdr;
                  T_RV_RET                    result;
                  UINT8                  nmb_of_partitions; //see note
                  T_GBI_PARTITION_INFO        *info_p;  //see note
                        }T_GBI_PARTITION_INFO_RSP_MSG;
```

Note: the members “nmb_of_partitions” and “info_p” are only valid if the “result” member has the value RV_OK. The “info_p” member is a pointer to an array of structures of type “T_GBI_MEDIA_INFO”. The array is “nmb_of_media” long.

The possible values of the result are :

id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not able to handle this request at this moment.
RV_INVALID_PARAMETER	The mb_id parameter is incorrect.
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_INTERNAL_ERROR	There was an internal error while executing the request.

Current restriction of use

None.

20.2.8 gbi_ subscribe_events

```
T_RV_RET gbi_subscribe_events(T_GBI_EVENTS     event,
                              T_RV_RETURN      return_path);
```

Description

This function allows the client to subscribe to certain events. A maximum of GBI_MAX_EVENT_SUBSCRIBERS can subscribe to a specific event. The function returns immediately after the parameters are verified. After this subscription, the client can expect to be notified asynchronous in the way specified in the return_path parameter.

Parameters

- **T_GBI_EVENTS**

A combination of events (logical OR) to which the client can subscribe. For example is media-removal.

- **T_RV_RETURN**

The return path of the client. The structure provides information about the way the driver must react asynchronous (whether to use a call-back principle or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_MEMORY_ERR	Allocation for messge buffer failed.
RV_INVALID_PARAMETER	Either Both Callback function and address id are absent or event supplied is incorrect
RVF_INTERNAL_ERR	Gpf_Send_msg failed.

Event Return

- **T_GBI_EVENT_MSG**

This event is the status sent at the end of the API.

Id	Definition
GBI_EVENT_MEDIA_NONE	No media present
GBI_EVENT_MEDIA_INSERT	Media has been inserted
GBI_EVENT_MEDIA_REMOVEAL	Media can been removed
GBI_EVENT_NAN_MEDIA_AVAILABLE	Nan media is available

Current restriction of use

None.

20.2.9 gbi_read_with_spare

```
T_RV_RET gbi_read_with_spare (UINT8          media_nmb,
                              UINT8          partition_nmb,
                              T_GBI_BLOCK    first_block,
                              T_GBI_BLOCK    number_of_blocks,
                              UINT32        *info_data_p
                              T_RV_RETURN    return_path)
```

Description

This function reads the spare area for a number of blocks. The blocks are consecutive and belong to the media and partition specified. The data is copied to the buffer, which is to be reserved by the client. The read starts from the first logical block first_block.The function returns immediately (non-blocking). Only the return path is verified immediately.Processing is done asynchronous and the result is returned through the return path

Parameters

- **media_nmb**

This identifies the media for which the command is intended. This number is provided when retrieving the overall partition table from the GBI.

- **partition_nmb**

This identifies the partition for which the command is intended. The partition must be located on the media specified by the media_nmb parameter. This number is provided when retrieving the overall partition table from the GBI .

- **first_block_nmb**

The first logical block number from where the information is requested.

- **number_of_blocks**

The number of logical blocks for which the information is to be read. The number may be from one to the last possible block number.

- **info_data_p**

This is a pointer to the buffer to which the block information is to be copied.

- **return_path**

This is the return path of the client. The structure provides information about the way the driver must react synchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RV_MEMORY_ERR	Allocation for messge buffer failed.
RV_INVALID_PARAMETER	Both Callback function and address id are absent .
RVF_INTERNAL_ERR	Gpf_Send_msg failed.

Event Return

- **T_GBI_READ_SPARE_DATA_RSP_MSG**

```
typedef struct {  T_RV_HDR              hdr;
                  T_RV_RET              result;
                } T_GBI_READ_SPARE_DATA_RSP_MSG;
```

The possible values of result are :

id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not able to handle this request at this moment.
RV_INVALID_PARAMETER	One or more parameters are invalid (*media_nmb, partition_nmb* is incorrect, *first_blo*ck is out of range, *number_of_blocks* is out of range, *info_data_p* pointer is a null pointer)
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_INTERNAL_ERROR	There was an internal error while executing the request.
RV_NOT_SUPPORTED	The driver does not support the spare data feature.

Current restriction of use

None.

20.2.10 gbi_ get_sw_version

```
UINT32 gbi_get_sw_version(void)
```

Description

This function returns the version of this service entity.

Parameters

None

Immediate Return

- **UINT32**

Returns version number.

Event Return

None

Current restriction of use

None.

Chapter 21 LLS

21.1 Introduction

This document provides an interface specification of the LLS(LED and Backlight control).

LLS is a separate task responsible for glowing of LEDs. LLS also controls the backlight of the handset.It's a independent task.

21.2 Interface description

21.2.1 lls_switch_on

```
T_RV_RET lls_switch_on(T_LLS_EQUIPMENT equipment_sort)
```

Description

This function can enable the LED or BACKLIGHT

Parameters

- **T_LLS_EQUIPMENT**

This parameter specifies the equipment to be switched on. Following are the values possible:
LLS_KEYPAD_LIGHT
LLS_BACKLIGHT
LLS_SUBPANEL_LIGHT

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RVF_INTERNAL_ERR	Environment block or parameter initialized is false
RV_INVALID_PARAMETER	Equipment value is not correct

Event Return

None

Current restriction of use

None.

21.2.2 lls_switch_off

```
T_RV_RET lls_switch_on(T_LLS_EQUIPMENT equipment_sort)
```

Description

This function can disable the LED or BACKLIGHT

Parameters

- **T_LLS_EQUIPMENT**

This parameter specifies the equipment to be switched on. Following are the values possible:
LLS_KEYPAD_LIGHT
LLS_BACKLIGHT
LLS_SUBPANEL_LIGHT

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RVF_OK	The API function successfully executed.
RVF_INTERNAL_ERR	Environment block or parameter initialized is false
RV_INVALID_PARAMETER	Equipment value is not correct

Event Return

None

Current restriction of use

None.

Chapter 22 MKS

22.1 Introduction

This chapter provides an interface specification of the MKS(Magic Key Sequence).
MKS allows special sequence to get configured using keypad .When the special sequence is pressed then message is sent to the subscriber.Each key sequence have a name and completion of sequence can be of two types : sequence completed or post sequence.

22.2 Interface description

22.2.1 mks_add_key_sequence

```
T_RV_RET mks_add_key_sequence(T_MKS_INFOS_KEY_SEQUENCE *
infos_key_sequence_p)
```

Description

Function initializes a magic key sequence.

Parameters

- **T_MKS_INFOS_KEY_SEQUENCE**

```
typedef struct {  char name[KPD_MAX_CHAR_NAME+1];
                  UINT8 nb_key_of_sequence;
                  T_KPD_VIRTUAL_KEY_ID  key_id[MKS_NB_MAX_OF_KEY_IN_KEY_SEQUENCE];
                  UINT8 completion_type ;
                  UINT8 nb_key_for_post_sequence;
                  T_RV_RETURN return_path;
                  } T_MKS_INFOS_KEY_SEQUENCE;
```

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function successfully executed.
RV_MEMORY_ERR	Memory allocation for message failed
RV_INVALID_PARAMETER	Actual parameters are not valid

Event Return

None

Current restriction of use

None.

22.2.2 mks_remove_key_sequence

```
T_RV_RET mks_remove_key_sequence(char name[KPD_MAX_CHAR_NAME+1]);
```

Description

Function removes an existing magic key sequence.

Parameters

- **name[KPD_MAX_CHAR_NAME + 1]**

Name of the key sequence to be removed.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function successfully executed.
RV_MEMORY_ERR	Memory allocation for message failed

Event Return

None

Current restriction of use

None.

Chapter 23 NAND

23.1 Introduction

This document describes the Riviera API for the NAND Block Manager entity (NAND_BM). This entity provides read and write access to the NAND-hardware through a logical block interface. Besides the low level access to the NAND-hardware the entity takes care of wear-levelling, bad block management and error correction (ECC).

The NAND_BM provides an asynchronous interface to the user. Both the read and write function allow multiple blocks to be read/written in a single request.

The API provides a logical block interface to the user. This means that the user accesses logical blocks which are translated to physical NAND-pages by the NAND_BM. The user will not be able to access physical NAND-pages directly.

Each logical block consists of a main area (512 bytes) and a spare area (16 bytes) according to the NAND-physical structure. The main area of a logical block maps to the main area of a NAND page and spare area of a logical block maps to the spare area of a NAND page. The NAND main area is completely available to the user. The NAND spare area however is only partly available to the user because it contains NAND_BM meta data as well (e.g. the ECC value). Note that a FAT file system is not using this spare area. However a NAND specific file system will use it to store administration data.

23.2 Interface description

23.2.1 nan_bm_read

```
T_RV_RET nand_bm_read(  UINT32          first_block,
                        UINT32          number_of_blocks,
                        UINT32          remainder_length,
                        UINT32          *buffer_p,
                        UINT32          *spare_buffer_p,
                        T_RV_RETURN     return_path)
```

Description
This function reads a number of logical blocks including spare data from NAND

Parameters

- **first_block**

The first logical block number from where the data is requested.

- **number_of_blocks**

The number of logical blocks to read. The number may be in range from one to the last possible block number.

- **remainder_length**

The number of bytes to read after the last whole block. A value of zero indicates no remainder.

- **buffer_p**

This is a pointer to the destination buffer to which the main data will be copied. Shall be set to NULL if not used.

- **spare_buffer_p**

This is a pointer to the destination buffer to which the spare data shall be copied. Shall be set to NULL if not used.

- **return_path**

This structure provides information about the way the driver must react asynchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	Invalid return path.
RV_NOT_READY	The entity is not able to handle this request at this moment (SWE not initialised).
RV_MEMORY_ERR	The entity has insufficient RAM resources to process the request.
RV_INTERNAL_ERR	There was an internal error and the request could not be processed.
RV_ECC_ERROR	The ECC correction failed. Data returned is corrupt.

Event Return

- **NAND_BM_READ_RSP_MSG**

This message is returned when the read request is processed. The message may indicate an error or completion of the read. In case of the completion of the read, the buffer holds the requested data. In case of error, the buffer contents is undefined.

Current restriction of use

None.

23.2.2 nand_bm_write

```
T_RV_RET nand_bm_write( UINT32          first_block,
                        UINT32          number_of_blocks,
                        UINT32          remainder_length,
                        UINT32          *buffer_p,
                        UINT32          *spare_buffer_p,
                        T_RV_RETURN     return_path)
```

Description

This function writes a number of logical blocks including spare data to NAND.

Parameters

- **first_block**

The first logical block number to where the data must be written.

- **number_of_blocks**

The number of logical blocks to write. The number may be in range from one to the last possible block number.

- **remainder_length**

The number of bytes to write after the last whole block is written. A value of zero indicates no remainder.

- **buffer_p**

A pointer to the source buffer from where the main data is to be copied. Shall be set to NULL if not used.

- **spare_buffer_p**

This is a pointer to the source buffer from where the spare data shall be copied. Shall be set to NULL if not used.

- **return_path**

This structure provides information about the way the driver must react asynchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	Invalid return path.
RV_NOT_READY	The entity is not able to handle this request at this moment (SWE not initialised).
RV_MEMORY_ERR	The entity has insufficient RAM resources to process the request.
RV_INTERNAL_ERR	There was an internal error and the request could not be processed.

Event Return

- **NAND_BM_WRITE_RSP_MSG**

This message is returned when the request is processed. The message may indicate an error or success when the write is completed. The data buffer needs to be unchanged until this response message is send.

Current restriction of use

None.

23.2.3 nan_bm_erase

```
T_RV_RET nand_bm_erase( UINT32          first_block,
                        UINT32          number_of_blocks,
                        T_RV_RETURN     return_path)
```

Description

This function erases a number of logical data blocks from the NAND (main+spare area). The erase starts from the first logical block first_block and continues for number_of_blocks blocks. After an erase the block data will be 0xFF.

.

Parameters

- **first_block**

The first logical block number to erase.

- **number_of_blocks**

The number of logical blocks to erase. The number may be from one to the last possible block number.

- **return_path**

This structure provides information about the way the driver must react asynchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	Invalid return path.
RV_NOT_READY	The driver is not able to handle this request at this moment (SWE not initialised).
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_INTERNAL_ERR	There was an internal error and the request could not be processed.

Event Return

- **NAND_BM_ERASE_RSP_MSG**

This message is returned when the request is processed. The message may indicate an error or success when the write is completed. The data buffer needs to be unchanged until this response message is send.

Current restriction of use

None.

23.2.4 nand_bm_flush

```
T_RV_RET nand_bm_flush(T_RV_RETURN  return_path)
```

Description

Flushes all cached data inside NAND_BM to the NAND-hardware. After receiving the response message of this function it is guaranteed that all data is synchronised to the NAND-hardware.

Parameters

- **return_path**

This structure provides information about the way the driver must react asynchronous (call-back pointer or a return message).

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	The *return_path* parameter is invalid.
RV_NOT_READY	The driver is not able to handle this request at this moment (SWE not initialised).
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_INTERNAL_ERR	There was an internal error and the request could not be processed.

Event Return

- **NAND_BM_FLUSH_RSP_MSG**

This message is returned when the request has been processed. The message may indicate an error or completion of the request. In case of the completion, all unwritten data has been written to the media.

Current restriction of use

None.

23.2.5 nand_bm_get_media_info

```
T_RV_RET nand_bm_get_media_info(T_NAND_BM_MEDIA_INFO *media_info)
```

Description

This function returns NAND-media info. Information returned in T_NAND_BM_MEDIA_INFO is:

variable	Description
nr_logical_blocks	Total number of logical blocks. Note that this value is smaller than the number of physical NAND pages in the device. This is because part of the NAND area is used to store meta data.
logical_block_size	Logical block size in bytes. Note that this excludes the spare size.
logical_block_spare_size	Spare size in bytes for each logical block. Will be 6 bytes

Parameters

- **media_info**

Pointer to T_NAND_BM_MEDIA_INFO structure which is allocated by the user.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	The API function was successfully executed.
RV_INVALID_PARAMETER	The *return_path* parameter is invalid.
RV_NOT_READY	The driver is not able to handle this request at this moment (SWE not initialised).

Event Return

None

Current restriction of use

None.

23.2.6 nand_bm_get_sw_version

```
UINT32 nand_bm_get_sw_version(void)
```

Description

This function returns the version of this service entity.

Parameters

None.

Immediate Return

- **UINT32**

Bit	ame	Function
[0-15]	BUILD	Build number
[16-23]	MINOR	Minor version number
[24-31]	MAJOR	Major version number

Event Return

None

Current restriction of use

None.

23.3 Message definition

One generic message type is defined called T_NAN_BM_MSG which is used for all request and response messages. The field used will depend on the request/response.

23.3.1 Request

Generally, a NAN_BM request is formed as follows:

```
m->os_hdr.msg_id = NAN_BM_xxx_REQ_MSG
m->rp = <return path>
```

Other fields may be defined depending on the message ID.

23.3.2 Response

The NAND driver will respond to a request with the following message:

```
m->os_hdr.msg_id = NAN_xxx_RSP_MSG
m->result = <   RV_OK |
                RV_INVALID_PARAMETER |
                RV_NOT_READY |
                RV_INTERNAL_ERR |
                RV_MEMORY_ERR >
```

23.3.3 NAND_BM_READ_REQ_MSG

The following fields shall be provided:

```
m->os_hdr.msg_id = NAN_BM_READ_REQ_MSG
m->rp = <return path>
m->first_block = <first block number>
m->number_of_blocks = < number of blocks>
m->remainder_length = <remainder length>
m->buffer_p = <source buffer>
m->spare_buffer_p = <spare buffer>
```

23.3.4 NAND_BM_READ_RSP_MSG

The NAN driver responds to a NAND_BM_READ_REQ_MSG event by returning a message with the following fields filled in:

```
m->os_hdr.msg_id = NAN_BM_READ_RSP_MSG
m->result = <result>
```

The possible values for *result* are:

Id	Definition
RV_OK	Success

RV_INVALID_PARAMETER	One ore more of the parameters is incorrect (first_block_nmb out of range, number_of_blocks out of range, remainder_length out of range, buffer_p is null pointer).
RV_NOT_READY	The driver is not able to handle this request at this moment (SWE not initialised).
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check if the media is removed or exchanged.
RV_ECC_ERROR	The ECC correction failed. Data returned is corrupt.

23.3.5 NAND_BM_WRITE_REQ_MSG

The client must supply the following fields for programming:

```
m->os_hdr.msg_id = NAN_BM_WRITE_REQ_MSG
m->rp = <return path>
m->first_block = <first block number>
m->number_of_blocks = < number of blocks>
m->remainder_length = <remainder length>
m->buffer_p = <source buffer>
m->spare_buffer_p = <spare buffer>
```

23.3.6 NAND_BM_WRITE_RSP_MSG

The NAN driver responds to a NAND_BM_WRITE_REQ_MSG event by returning a message with the following field filled in:

```
m->os_hdr.msg_id = NAN_BM_WRITE_RSP_MSG
m->result = <result>
```

The possible values for *result* are:

Id	Definition
RV_OK	Success
RV_INVALID_PARAMETER	One ore more of the parameters is incorrect (first_block_nmb out of range, number_of_blocks out of range, remainder_length out of range, buffer_p is null pointer).
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_NOT_READY	The driver is not able to handle this request at this moment (SWE not initialised).
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check if the media is removed or exchanged.

23.3.7 NAND_BM_ERASE_REQ_MSG

The following fields shall be provided:

```
m->os_hdr.msg_id = NAN_BM_ERASE_REQ_MSG
m->rp = <return path>
m->first_block = <first block number>
m->number_of_blocks = < number of blocks>
```

23.3.8 NAND_BM_ERASE_RSP_MSG

```
m->os_hdr.msg_id = NAN_BM_ERASE_RSP_MSG
m->result = <result>
```

The possible values for *result* are:

Id	Definition
RV_OK	Success
RV_INVALID_PARAMETER	One ore more of the parameters is incorrect. (Invalid *first_block_nmb* out of range, *number_of_blocks* out of range,).
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_NOT_READY	The driver is not able to handle this request at this moment (SWE not initialised).

RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check if the media is removed or exchanged.

23.3.9 NAND_BM_FLUSH_REQ_MSG

The following fields shall be provided:

```
m->os_hdr.msg_id = NAN_BM_FLUSH_REQ_MSG
m->rp = <return path>
```

23.3.10 NAND_BM_FLUSH_RSP_MSG

```
m->os_hdr.msg_id = NAN_BM_FLUSH_RSP_MSG
m->result = <result>
```

The possible values for *result* are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not able to handle this request at this moment.
RV_MEMORY_ERR	The driver has insufficient RAM resources to process the request.
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check if the media is removed or exchanged.

23.4 Types definition

23.4.1 T_NAND_BM_MEDIA_INFO

This structure defines the information for the NAND-media.

```
typedef struct{
            UINT32              nr_logical_blocks;
            UINT32              logical_block_size;
            UINT32              logical_block_spare_size;
            } T_NAND_BM_MEDIA_INFO;
```

23.5 Configuration Items

23.5.1 Feature settings

23.5.1.1 NAND_BM_ECC

Defines if ECC shall be enabled or not. To disable ECC the macro shall be commented out. Note that enabling ECC will decrease the read/write performance.

```
#define NAN_BM_ECC_ENABLED.
```

23.5.2 Hardware settings

23.5.2.1 Memory size

23.5.2.1.1 NAN_BM_MAX_CHIP_SELECT

Defines the chip select which shall be used.

```
#define NAN_BM_MAX_CHIP_SELECT      0
```

23.5.2.1.2 NAN_BM_FLASH_NOF_BLOCKS

Defines the number of NAND-blocks in the device.

```
#define NAN_BM_FLASH_NOF_BLOCKS     4096        //64MB device
```

23.5.2.1.3 NAN_BM_FLASH_PAGES_PER_BLOCK

Defines the number of NAND-pages per NAND-block.

```
#define NAN_BM_FLASH_PAGES_PER_BLOCK 32         //16kB/block
```

23.5.2.1.4 NAN_BM_FLASH_PAGE_SIZE

Defines the number of bytes per NAND-page (main+spare).

```
#define NAN_BM_FLASH_PAGE_SIZE 528              //bytes
```

23.5.2.1.5 NAN_BM_FLASH_SPARE_OFFSET

Defines the offset in NAND-page for the spare area (or the size of the main area).

```
#define NAN_BM_FLASH_SPARE_OFFSET 512           //bytes
```

23.5.2.2 Timing

The NAND flash controller contains a number of timing settings which can be configured at compile time.

23.5.2.2.1 NAN_BM_FLASH_CLK_DIV

Clock divider value for the prescaler, 1 – 15:

```
#define NAN_BM_FLASH_CLK_DIV 4
```

23.5.2.2.2 NAN_BM_DUMMY_CYCLE

NAND Busy delay, range 0-15 cycles:

```
#define NAN_BM_DUMMY_CYCLE 15
```

NOTE: this value is scaled by the pre-scaler.

Chapter 24 SIM

24.1 Introduction

This document provides an interface specification of the SIM Driver entity. The purpose of the SIM Driver entity is to provide a low level access (L1) to the SIM Card content.

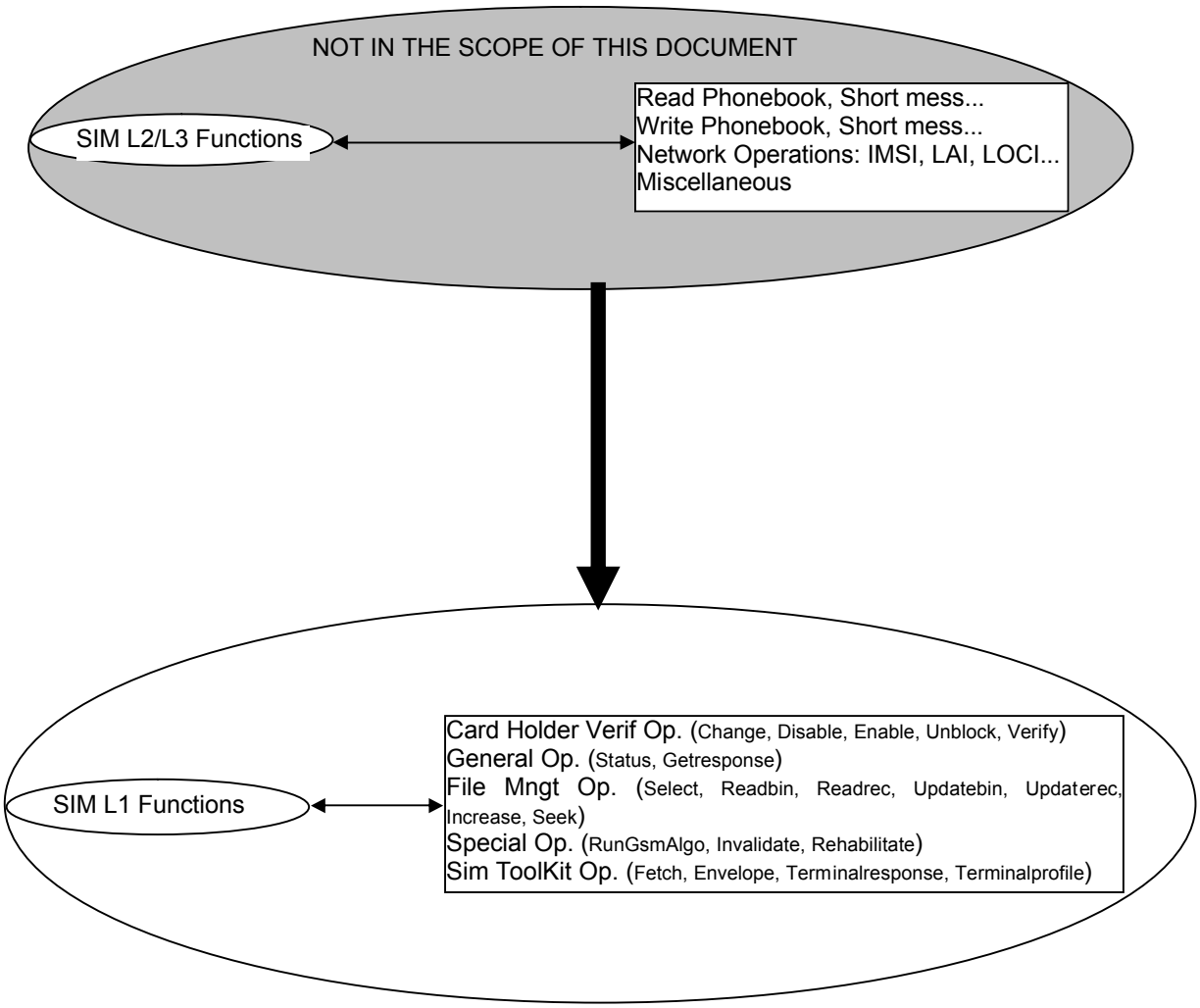
The Driver API is based on the ETSI/3GPP 11.11 and ISO/CEI 7816-3 commands. For detailed requests, refer to them.

ATTENTION :

A) The actual version of the SIM Driver is not compliant with the Riviera Technology.

B) The caller must plan 10 machine words in it's stack in order to prevent Stack overflow.

C) The SIM Driver is not protected against concurrential accesses.



24.2 SIM Driver Services functions

24.2.1 Initializing and cleaning up operations

For starting the SIM Driver you need three functions. They are:

SIM_Register for register yours callbacks for insertion and removal of the Card,
SIM_Initialize for initialize internal structures and the SIM Driver timer,
SIM_Reset for hardware power on of the interface and the Card.

The above chronology is to be fulfilled. At the end of these three steps, the SIM Card is accessible to requests

For cleaning up the SIM Driver you need only one function viz SIM_PowerOff for cleaning all the registers and the Card.

24.2.1.1 SIM_Register

```
SYS_UWORD16 SIM_Register (void (Insert(SIM_CARD *cP)),
                          void (Remove(void)))
```

Description

This function allows a entity to register the callbacks functions managing the hardware insertion or removal of the Card. This function should be called several times during a session, but usually it is called only one time in a session.

Parameters

- **Insert**

Function name of the callback for insertion.

- **Remove**

Function name of the callback for removal.

Return value

SIM_OK

24.2.1.2 SIM_Initialize

```
void SIM_Initialize(void)
```

Description

This function initializes the internal structures and set the SIM Driver timer. Shall be called only once by GSM/GPRS session.

Parameters

None.

Return value

None

24.2.1.3 SIM_Reset

```
SYS_UWORD16 SIM_Reset(SIM_CARD *cP)
```

Description

This function performs the hardware initialization of the Interface and then the SIM Card.
The main steps of this function are:

- Sets the right values in the right registers depending on the hardware used (voltage etc)
- Manual start procedure
- Dynamic ATR Procedure
- PTS Procedure (speed operations)
- Voltage switching (depending on the behavior of the interface and the Card)
- File characteristics detection (clock stop, speed for authentication procedure).

This function shall be called only one time during a GSM/GPRS session.

Parameters

- **cP**

Pointer on some characteristics of the Card (ATR, ATR size, convention type)

Immediate Return

Only internal values, see return value chapter.

24.2.1.4 SIM_PowerOff

```
void SIM_PowerOff(void)
```

Description

This function shutdown all the signals between the Interface and the Card, clean the registers of the Interface and delete the SIM Driver timer.

Parameters

None.

Immediate Return

None.

24.2.2 Card Holder Verification Operations

24.2.2.1 SIM_VerifyCHV

```
SYS_UWORD16 SIM_VerifyCHV(SYS_UWORD8 *result, SYS_UWORD8 *dat,
                          SYS_UWORD8 chvType, SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to verify his CHV (level 1 or more), by sending a command to the SIM Card. It execute the VERIFY CHV command specified by ETSI 11.11.

Parameters

- **result**

Pointer on the bytes received

- **dat**

Pointer on the CHV to verify. It must be composed of 8 bytes.

- **chvType**

Type of CHV. Used for parameter P2 of a SIM command

- **rcvSize**

Pointer on the number of bytes received.

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11 : Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.2.2 SIM_ChangeCHV

```
SYS_UWORD16 SIM_ChangeCHV(SYS_UWORD8 *result, SYS_UWORD8 *oldChv,
                          SYS_UWORD8 *newChv, SYS_UWORD8 chvType,
                          SYS_UWORD16 *lP)
```

Description

This function allows the user to change his CHV (level 1 or more), by sending a command to the SIM Card. It execute the CHANGE CHV command specified by ETSI 11.11.

Parameters

- **result**

Pointer on the bytes received

- **oldChv**

Pointer on the old CHV to use. It must be composed of 8 bytes

- **newChv**

Pointer on the new CHV to use. It must be composed of 8 bytes

- **chvType**

Type of CHV. Used for parameter P2 of a SIM command

- **lP**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11 : Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.2.3 SIM_DisableCHV

```
SYS_UWORD16 SIM_DisableCHV(SYS_UWORD8 *result, SYS_UWORD8 *dat,
                           SYS_UWORD16 *lP)
```

Description

This function allows the user to disable his CHV1 typed required, by sending a command to the SIM Card. It executes the DISABLE CHV type 1 command specified by ETSI 11.11.

Parameters

- **result**

Pointer on the bytes received

- **dat**

Pointer on the current CHV1. It must be composed of 8 bytes

- **lP**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11 : Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.2.4 SIM_EnableCHV

```
SYS_UWORD16 SIM_EnableCHV(SYS_UWORD8 *result, SYS_UWORD8 *dat,
                          SYS_UWORD16 *lP)
```

Description

This function allows the user to enable his CHV1 typed required, by sending a command to the SIM Card. It executes the ENABLE CHV type 1 command specified by ETSI 11.11.

Parameters

- **result**

Pointer on the bytes received

- **dat**

Pointer on the current CHV1. It must be composed of 8 bytes

- **lP**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11 : Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.2.5 SIM_UnblockCHV

```
SYS_UWORD16 SIM_UnblockCHV(SYS_UWORD8 *result, SYS_UWORD8 *unblockChv,
                           SYS_UWORD8 *newChv,  SYS_UWORD8 chvType,
                           SYS_UWORD16 *lP)
```

Description

This function allows the user to unblock his CHV, by sending a command to the SIM Card. When using a wrong CHV code three times in succession, the CHV becomes unusable and by the way the Card too. It is possible to unblock the CHV (and the Card) by using this function. It executes the UNBLOCK CHV command specified by ETSI 11.11.

Parameters

- **result**

Pointer on the bytes received

- **unblockChv**

Pointer on the unblock CHV to use. It must be composed of 8 bytes

- **newChv**

Pointer on the new CHV to use. It must be composed of 8 bytes

- **chvType**

Type of CHV. Used for parameter P2 of a SIM command

- **IP**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11 : Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.3 General Operations

24.2.3.1 SIM_Status

```
SYS_UWORD16 SIM_Status(SYS_UWORD8 *dat, SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to get several generals informations about the current position on the Card. In a second way, it gives the ability to Check if the Card want to send a SimToolKit request to the mobile. It execute the STATUS command specified by ETSI 11.11.

Parameters

- **dat**

Pointer on the bytes received

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11 : Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.3.2 SIM_Status_Extended

```
SYS_UWORD16 SIM_Status_Extended(SYS_UWORD8 *dat, SYS_UWORD16 len,
                                SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to get several generals informations about the current position on the Card. This is a derivation of the SIM_Status function; It allows the user not to read all (22 bytes) the bytes of the header of the current position, but only the number required. In a second way, it gives the ability to Check if the Card wants to send a SimToolKit request to the mobile. It executes the STATUS command specified by ETSI 11.11.

Parameters

- **dat**

Pointer on the bytes received

- **len**

Number of bytes to read. Up to 22 bytes

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11 : Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.3.3 SIM_GetResponse

```
SYS_UWORD16 SIM_GetResponse(SYS_UWORD8 *dat, SYS_UWORD16 len,
                            SYS_UWORD16 *rcvSize)
```

Description

This function allows the user obtain the answers to a previous command. It executes the GET RESPONSE command specified by ETSI 11.11.

Parameters

- **dat**

Pointer on the bytes received

- **len**

Number of bytes to return. (used in parameter P3 of the SIM command)

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11 : Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.4 File Management Operations

24.2.4.1 SIM_Select

```
SYS_UWORD16 SIM_Select(SYS_UWORD16 id, SYS_UWORD8 *dat,
                       SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to position the logical internals pointers (R, W) of the SIM Card on the required directory or file. It execute the SELECT command specified by ETSI 11.11.

Parameters

- **id**

ETSI Identity of the file selected

- **dat**

Pointer on the bytes received

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.4.2 SIM_ReadBinary

```
SYS_UWORD16 SIM_ReadBinary(SYS_UWORD8 *dat, SYS_UWORD16 offset,
                           SYS_UWORD16 len, SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to read a Binary file previously selected. It executes the READ BINARY command specified by ETSI 11.11.

Parameters

- **dat**

Pointer on the bytes received

- **offset**

Offset from where you start to read the bytes in the file. (Used in parameters P1 and P2 of the SIM Command)

- **len**

Number of bytes to read (Used in parameter P3 of the SIM command)

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.4.3 SIM_UpdateBinary

```
SYS_UWORD16 SIM_UpdateBinary(SYS_UWORD8 *result, SYS_UWORD8 *dat,
                             SYS_UWORD16 offset, SYS_UWORD16 len,
                             SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to update a Binary file previously selected. It executes the UPDATE BINARY command specified by ETSI 11.11.

Parameters

- **result**

Pointer on the bytes received

- **dat**

Pointer on the bytes to write limited by *len*

- **offset**

Offset from where you start to update the bytes in the file. (Used in parameters P1 and P2 of the SIM Command)

- **len**

Number of bytes to write (Used in parameter P3 of the SIM command)

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.4.4 SIM_ReadRecord

```
SYS_UWORD16 SIM_ReadRecord(SYS_UWORD8 *dat, SYS_UWORD8 mode,
                           SYS_UWORD8 recNum, SYS_UWORD16 len,
                           SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to read a Record Structured file (linear Fixed, or cyclic) previously selected. It executes the READ RECORD command specified by ETSI 11.11.

Parameters

- **dat**

Pointer on the bytes received

- **mode**

Specify the mode chosen to read this record. (Used in parameter P2 of the SIM command)

- **recNum**

Specify the record to read. (Used in parameter P1 of the SIM command)

- **len**

Number of bytes to read (Used in parameter P3 of the SIM command)

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.4.5 SIM_UpdateRecord

```
SYS_UWORD16 SIM_UpdateRecord(SYS_UWORD8 *result, SYS_UWORD8 *dat,
                             SYS_UWORD8 mode, SYS_UWORD8 recNum,
                             SYS_UWORD16 len, SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to update a Record Structured file (linear Fixed, or cyclic) previously selected. It executes the UPDATE RECORD command specified by ETSI 11.11.

Parameters

- **result**

Pointer on the bytes received

- **dat**

Pointer on the bytes to write limited by *len*

- **mode**

Specify the mode chosen to update this record. (Used in parameter P2 of the SIM command)

- **recNum**

Specify the record to update. (Used in parameter P1 of the SIM command)

- **len**

Number of bytes to write (Used in parameter P3 of the SIM command)

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.4.6 SIM_Seek

```
SYS_UWORD16 SIM_Seek(SYS_UWORD8 *result, SYS_UWORD8 *dat, SYS_UWORD8 mode,
                     SYS_UWORD16 len, SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to be positioned directly on the required record number of a Record Structured file (linear Fixed, or cyclic) previously selected. It executes the SEEK command specified by ETSI 11.11.

Parameters

- **result**

Pointer on the bytes received

- **dat**

Pointer on the pattern to seek

- **mode**

Specify the mode chosen to seek a field. (Used in parameter P2 of the SIM command)

- **len**

Number of bytes to seek (Used in parameter P3 of the SIM command)

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.4.7 SIM_Increase

```
SYS_UWORD16 SIM_Increase(SYS_UWORD8 *result, SYS_UWORD8 *dat,
                         SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to be positioned directly on the required record number of a Record Structured file (linear Fixed, or cyclic) previously selected. It executes the SEEK command specified by ETSI 11.11.

Parameters

- **result**

Pointer on the bytes received

- **dat**

Pointer on the bytes composing the value to use to increase the original field

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.5 Special Operations

24.2.5.1 SIM_RunGSMAlgo

```
SYS_UWORD16 SIM_RunGSMAlgo(SYS_UWORD8 *result, SYS_UWORD8 *dat,
                           SYS_UWORD16 *rcvSize)
```

Description

This function allows the user execute the A3-A8 algorithm (specified by ETSI 11.11). It executes the RUN GSM ALGO command specified by ETSI 11.11.

Parameters

- **result**

Pointer on the bytes received

- **dat**

Pointer on the bytes to send to the SIM Card; There are 16 bytes to send

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.5.2 SIM_Invalidate

```
SYS_UWORD16 SIM_Invalidate(SYS_UWORD8 *rP, SYS_UWORD16 *lP)
```

Description

This function allows the user to INVALIDATE a file (set it as unusable). It executes the INVALIDATE command specified by ETSI 11.11.

Parameters

- **rP**

Pointer on the bytes received

- **lP**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.5.3 SIM_Rehabilitate

```
SYS_UWORD16 SIM_Rehabilitate(SYS_UWORD8 *rP, SYS_UWORD16 *lP)
```

Description

This function allows the user to REHABILITATE a file (set it as usable). It executes the REHABILITATE command specified by ETSI 11.11.

Parameters

- **rP**

Pointer on the bytes received

- **lP**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.6 SimToolKit Operations

24.2.6.1 SIM_TerminalProfile

```
SYS_UWORD16 SIM_TerminalProfile(SYS_UWORD8 *result, SYS_UWORD8 *dat,
                                SYS_UWORD16 len, SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to send the capabilities of the mobile concerning the SIM Application ToolKit (specified by ETSI 11.11 and 11.14). It executes the TERMINAL PROFILE command specified by ETSI 11.11 and 11.14.

Parameters

- **result**

Pointer on the bytes received

- **dat**

Pointer on the bytes composing the profile, to send to the SIM Card

- **len**

Number of bytes to send (Used in parameter P3 of the SIM command)

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.6.2 SIM_Fetch

```
SYS_UWORD16 SIM_Fetch(SYS_UWORD8 *result, SYS_UWORD16 len,
                      SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to send an Application ToolKit command from the SIM Card to the ME, or keypad, or display or earpiece, or network, or another device. It executes the FETCH command specified by ETSI 11.11, and ETSI 11.14.

Parameters

- **result**

Pointer on the bytes received

- **len**

Number of bytes to send (Used in parameter P3 of the SIM command)

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.6.3 SIM_TerminalResponse

```
SYS_UWORD16 SIM_TerminalResponse(SYS_UWORD8 *result, SYS_UWORD8 *dat,
                                 SYS_UWORD16 len, SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to transfer from ME to SIM the response to a previously fetched SIM Application ToolKit command. It executes the TERMINAL RESPONSE command specified by ETSI 11.11, and ETSI 11.14.

Parameters

- **result**

Pointer on the bytes received

- **dat**

Pointer on the bytes composing the response, to send to the SIM Card

- **len**

Number of bytes to send (Used in parameter P3 of the SIM command)

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.6.4 SIM_Envelope

```
SYS_UWORD16 SIM_Envelope(SYS_UWORD8 *result, SYS_UWORD8 *dat,
                         SYS_UWORD16 len, SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to transfer data to the SIM Application ToolKit in the SIM. It executes the ENVELOPE command specified by ETSI 11.11, and ETSI 11.14.

Parameters

- **result**

Pointer on the bytes received

- **dat**

Pointer on the bytes composing the data, to send to the SIM Card

- **len**

Number of bytes to send (Used in parameter P3 of the SIM command)

- **rcvSize**

Pointer on the number of bytes received

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11: Command versus possible status responses) and internal return value (see Return value Chapter).

24.2.7 Transparent Data Exchange

24.2.7.1 SIM_XchTPDU

```
SYS_UWORD16 SIM_XchTPDU(SYS_UWORD8 *dat, SYS_UWORD16 trxLen,
                        SYS_UWORD8 *result, SYS_UWORD16 rcvLen,
                        SYS_UWORD16 *rcvSize)
```

Description

This function allows the user to transparently exchange command and responses in TPDU format with the SIM. The sent data must at least contain a valid SIM command according to GSM 11.11 or ISO 7816-3. Response data is only returned, when *rcvLen* is non-zero. The function shall be able return less data as expected by the value given in *rcvLen*. The SIM status code is not part of the received data, it is given by the function's return value.

Parameters

- **dat**

Pointer on the bytes to be sent limited by *trxLen* (at least a valid SIM Command, extended by up to 255 bytes of data)

- **trxLen**

Length of the bytes to be sent

- **result**

Pointer on the bytes (excluding SIM Status Code SW1/SW2) received

- **rcvLen**

Number of bytes expected to be received (excluding SIM Status Code SW1/SW2)

- **rcvSize**

Pointer on the number of bytes received (excluding SIM Status Code SW1/SW2)

Immediate Return

Possible values are SW1 / SW2 (see ETSI 11.11 and ISO 7816-3: Command versus possible status responses) and internal return value (see Return value Chapter).

24.3 SIM Driver Return Values

Size: 16 bits.

Label	Values	Description
SIM_OK	0	Operation Successful
SIM_ERR_NOCARD	1	SIM card extraction indication
SIM_ERR_NOINT	2	Not used
SIM_ERR_NATR	3	problem with ATR reception
SIM_ERR_READ	4	Problem during reception of bytes
SIM_ERR_XMIT	5	Problem during transmission of bytes
SIM_ERR_OVF	6	Overflow of the FIFO
SIM_ERR_LEN	7	the answer is not corresponding to a correct answer of T=0 protocol
SIM_ERR_CARDREJECT	8	Card not acceptable for protocol T=0
SIM_ERR_WAIT	9	Indicates SW timer overflow
SIM_ERR_ABNORMAL_CASE1	10	abnormal case of the asynchronous state machine
SIM_ERR_ABNORMAL_CASE2	11	abnormal case of the synchronous state machine
SIM_ERR_BUFF_OVERFL	12	SIM copy exceeds 255 characters
SIM_ERR_HARDWARE_FAIL	13	Power on failure of the interface (valid for only for IOTA)
SIM_ERR_RETRY_FAILURE	14	Software retry failed

24.4 Type and Other Definitions

These type definitions are used at the driver interface:

```
typedef unsigned char  SYS_BOOL;

typedef unsigned char  SYS_UWORD8;
typedef signed   char  SYS_WORD8;

typedef unsigned short SYS_UWORD16;
typedef          short SYS_WORD16;

typedef unsigned long  SYS_UWORD32;
typedef          long  SYS_WORD32;

typedef struct
{
    SYS_UWORD8    Inverse;
    SYS_UWORD8    AtrSize;
    SYS_UWORD8    AtrData[MAX_ATR_SIZE];
} SIM_CARD;
```

These constant definitions are used at the driver interface:

```
#define MAX_ATR_SIZE          33
```

Chapter 25 DAR

25.1 Introduction

This document describes the APIs of DAR (Diagnistics And Recovery) module.

25.2 Interface Description

In this paragraph the interface of the DAR module is described.

25.2.1 Recovery Functions Definitions

25.2.1.1 dar_recovery_get_status

```
T_RV_RET dar_recovery_get_status(T_DAR_RECOVERY_STATUS* status)
```

Description

This function is called by the MMI at the beginning of the procedure, in order to get the status of the last reset of the system.

Parameters

- **status**

DAR recovery status is stored in *status.

Immediate Return

- **T_RV_RET**

This function always returns RV_OK.

25.2.1.2 dar_recovery_config

```
T_RV_RET dar_recovery_config(T_RV_RET (*dar_store_recovery_data)
                                      (T_DAR_BUFFER buffer_p,
                                       UINT16 length))
```

Description

This function is used to store a callback function that will be called by the recovery system when a recovery procedure has been initiated.

Parameters

- **dar_store_recovery_data**

dar callback function

Immediate Return

- **T_RV_RET**

This function always returns RV_OK.

25.2.1.3 dar_get_recovery_data

```
T_RV_RET dar_get_recovery_data( T_DAR_BUFFER buffer_p, UINT16 length )
```

Description

This function is used to retrieve data that have been stored in the buffer just before a reset.

Parameters

- **`buffer_p`**

The buffer in whom important data have been stored before the reset

- **`length`**

The length of the buffer

Immediate Return

- **T_RV_RET**

This function always returns RV_OK.

25.2.2 Watchdog Functions Definitions

25.2.2.1 dar_start_watchdog_timer

```
T_RV_RET dar_start_watchdog_timer(UINT16 timer_expiration_value)
```

Description

This function uses the timer as a general purpose timer instead of Watchdog. It loads the timer, starts it and then unmasks the interrupt.

Parameters

- **`timer_expiration_value`**

Timer's interval in milliseconds before the timer expires.

Immediate Return

- **T_RV_RET**

This function always returns RV_OK.

25.2.2.2 dar_reload_watchdog_timer

```
T_RV_RET dar_reload_watchdog_time(void)
```

Description

This function is used to maintain the timer in reloading it periodically before it expires.

Parameters

None

Immediate Return

- **T_RV_RET**

This function always returns RV_OK.

25.2.2.3 dar_stop_watchdog_timer

```
T_RV_RET dar_stop_watchdog_timer(void)
```

Description

This function stops the timer used as a general purpose timer instead of watchdog.

Parameters

None

Immediate Return

- **T_RV_RET**

This function always returns RV_OK.

25.2.3 Reset Functions Definition

25.2.3.1 dar_reset_system

```
T_RV_RET dar_reset_system(void)
```

Description

This function can be used to reset the system voluntarily.

Parameters

None

Immediate Return

- **T_RV_RET**

This function always returns RV_OK.

25.2.4 Diagnose Functions Definition

25.2.4.1 dar_diagnose_swe_filter

```
T_RV_RET dar_diagnose_swe_filter( T_RVM_USE_ID dar_use_id,
                                  T_DAR_LEVEL dar_level)
```

Description

This function is called to configure the Diagnose filtering. It allows to determine what Software Entity (dar_use_id) wants to use the Diagnose and allows to indicate the level threshold of the diagnose messages (Warning or Debug).

Parameters

- **dar_use_id**

The dar use id.

- **dar_level**

The dar level.

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_NOT_READY	DAR Entity is not running
RV_NOT_SUPPORTED	Insufficient resources
RV_MEMORY_ERR	Not enough memory

25.2.4.2 dar_diagnose_write

```
T_RV_RET dar_diagnose_write( T_DAR_INFO *buffer_p,
                             T_DAR_FORMAT format,
                             T_DAR_LEVEL diagnose_info_level,
                             T_RVM_USE_ID dar_use_id )
```

Description

This function is called to store diagnose data in RAM buffer.

Parameters

- **buffer_p**

Pointer to the message to store.

- **format**

Data Format (the Binary format is not supported).

- **diagnose_info_level**

Data level

- **dar_use_id**

Data Use ID

Immediate Return

- **T_RV_RET**

This function always returns RV_OK.

25.2.4.3 dar_diagnose_generate_emergency

```
T_RV_RET dar_diagnose_generate_emergency( T_DAR_INFO *buffer_p,
                                          T_DAR_FORMAT format,
                                          T_RVM_USE_ID dar_use_id )
```

Description

This function is called to store diagnose data in RAM buffer when an emergency has been detected and goes to emergency (automatic reset).

Parameters

- **buffer_p**

Pointer to the message to store.

- **`format`**

Data Format (the Binary format is not supported).

- **`dar_use_id`**

Data Use ID

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	The API function was successfully executed.
RV_NOT_READY	DAR entity is not running

25.3 Message Definition

The messages used in the DAR module are given below:

25.3.1 Diagnose Messages

25.3.1.1 DAR_FILTER_REQ

This message is used to diagnose filter and non-filter messages. The following fields are defined for this message:

msg_p->os_hdr.msg_id = DAR_FILTER_REQ

msg_p->os_hdr.src_addr_id = <dar address id>

msg_p->use_msg_parameter.group_nb = <group number>
msg_p->use_msg_parameter.mask = <mask>
msg_p->use_msg_parameter.level = <dar_level>

25.3.1.2 DAR_WRITE_REQ

This is the diagnose write message. The following fields aredefined:

msg_p->os_hdr.msg_id = DAR_WRITE_REQ
msg_p->os_hdr.src_addr_id = <dar address id>
msg_p->data_write.char_p = <buffer>
msg_p->data_write.data_format = <format>
msg_p->data_write.level = <diagnose info level>
msg_p->data_write.use_id.group_nb = <group number>
msg_p->data_write.use_id.mask = <mask>

25.4 Type Definition

25.4.1 T_DAR_RECOVERY_STATUS

typedef UINT16 T_DAR_RECOVERY_STATUS;

25.4.2 T_DAR_STATE

```
typedef INT8 T_DAR_STATE;
```

25.4.3 T_DAR_FORMAT

```
typedef INT8 T_DAR_FORMAT;
```

25.4.4 T_DAR_LEVEL

```
typedef UINT8 T_DAR_LEVEL;
```

25.4.5 T_DAR_BUFFER

```
typedef UINT8* T_DAR_BUFFER;
```

25.4.6 T_DAR_INFO

```
typedef char T_DAR_INFO;
```

25.4.7 T_DAR_USE_ID

```
typedef struct
    {
      UINT16 group_nb;
      UINT16 mask;
    }T_DAR_USE_ID;
```

25.4.8 T_DAR_STATUS

```
typedef struct
    {
      T_RV_HDR   os_hdr;
      INT8       status;
    } T_DAR_STATUS;
```

25.4.9 T_DAR_RECOVERY_CONFIG

```
typedef struct{
      UINT16          msg_id;              /* id of the message     */
      T_DAR_BUFFER    buffer_p;            /* pointer on the buffer */
      UINT8           length;              /* buffer length         */
    } T_DAR_RECOVERY_CONFIG;
```

25.4.10 T_DAR_FILTER_START

```
typedef struct
    {
      T_RV_HDR            os_hdr;
      T_DAR_MSG_PARAM     use_msg_parameter;
      T_RV_RETURN         return_path;
    }  T_DAR_FILTER_START;
```

25.4.11 T_DAR_ WRITE_START

```
 typedef struct
    {
      T_RV_HDR          os_hdr;
```

```
    T_DAR_WRITE        data_write;
    T_RV_RETURN        return_path;
} T_DAR_WRITE_START;
```

Chapter 26 AUDIO

26.1 Introduction

This document provides an interface specification of the AUDIO entity. The purpose of the AUDIO entity is to provide an abstraction layer to SW developers in order to access the audio services available on the platform.

Voice Memo, Voice Recognition, Melody generation, Key Beep generation, Tones generation, audio mode configuration and speaker volume are part of the services provided by the AUDIO entity.

26.2 Overview

26.2.1 Generality

The AUDIO services provided by the AUDIO entity can be accessed by any entity running on the mobile. Several entities can use the audio services at the same time. Note that for commodity reasons, the entity using the AUDIO services will be called MMI in the rest of the document.

All the services provided by the audio entity can be accessed via direct function call. These functions are listed in this document. The AUDIO entity use the return mechanism to provide information back to the MMI.

26.2.2 Return Mechanism

All the functions return an immediate value, providing information on the success or the failure of the function call. In some (most of the) cases, extra processing time is needed to perform the action requested when calling the function. In this case, the function is exit and later on, one or several EVENTs are sent back by the AUDIO entity. Note that for commodity, all the events are always mentioned in upper case.

The AUDIO entity use the EVENT format and the return path method defined in Riviera Environment. Basically, in order to send information back, the AUDIO entity sends EVENTs to the MMI. An event is a buffer, with a header, common to any EVENT, and a custom field related to the EVENT. The header is a C structure, containing the *opcode* field. This field contains the unique opcode of the EVENT and is the only way to know which kind of EVENT has been received. Based on this value, the MMI can re-cast the buffer and access to custom information related to the EVENT.

MMI have two ways to get access to the EVENTS: Call back functions or message posted in its mailbox.

A call back function is a function name, provided by MMI as a parameter and which will be called by the AUDIO SW when a EVENT is available. When a callback function is defined, it is always the callback function mechanism that is used to return EVENTs to the MMI.
But for more efficient implementation, it also possible to directly post a message into the MMI mailbox. In this case, the task id and mailbox id of the MMI must be provided to the AUDIO SW entity.

For every audio service, the MMI can define which return mechanism should be used. For that purpose, it must provide a *return_path*. The generic *return_path* type is a C structure, defined as:

```
/* unique ADDRess IDentifier of a SWE */
typedef UINT16 T_RVF_ADDR_ID;

typedef struct
{
   T_RVF_ADDR_ID addr_id;
   void          (*callback_func)(void *);
} T_RV_RETURN;
```

26.3 Audio-Modem Incompatibilities

Due to the share of the CPU load and of the memory in DSP, some audio task can't run with certain modem state. For instance, the audio MCU software doesn't manage this constraint. Therefore any user of the audio services described in this document must follow the audio-modem incompatibilities described in the table below.

In case of incompatibilities, the user of the audio must stop as soon as possible the audio before to enter to a modem state incompatible with the current audio task.

26.3.1 DSP codes <= 33

26.3.2 DSP code 34

Feature/Modem	0x3416, 0x4180									
	GSM		Dedicated mode				GPRS		GSM->GPRS	GPRS->GSM
	Idle	SMS	FACCH	Speech	TCH/Data	IDS	Idle	Transfer		
Keybeep	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Tones	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Melody_E1	✓	✓	✓	✓			✓	✓	✓	✓
Voice Memo	✓	✓	✓	✓	✓		✓	✓	✓	✓
AMR MMS	✓	✓	✓				✓	✓	✓	✓
Speech Reco	✓		✓				✓	✓	✓	✓
TTY				✓						
FIR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AEC				✓						
Audio Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TIDE/DCO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

26.3.3 DSP code 35

Feature/Modem	GSM		Dedicated non AMR					Dedicated AMR		GPRS		GSM->GPRS	GPRS->GSM
	Idle	SMS	FACCH	Ringer	Speech	TCH/Data	IDS	Ringer	Speech	Idle	Transfer		
Keybeep	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Tones	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Melody_E1	✓	✓	✓	✓	✓			✓		✓	✓	✓	✓
Voice Memo	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
AMR MMS	✓	✓	✓	✓				✓		✓	✓	✓	✓
Speech Reco	✓		✓							✓	✓	✓	✓
TTY					✓				✓				
FIR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AEC					✓				✓				
Audio Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TIDE/DCO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

26.3.4 DSP code 36

Feature/Modem	0x3606, 0x6180												
	GSM		Dedicated non AMR					Dedicated AMR		GPRS		GSM->GPRS	GPRS->GSM
	Idle	SMS	FACCH	Ringer	Speech	TCH/Data	IDS	Ringer	Speech	Idle	Transfer		
Keybeep	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Tones	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Melody_E1	✓	✓	✓	✓	✓			✓		✓	✓	✓	✓
Melody_E2	✓	✓	✓	✓				✓		✓	✓	✓	✓
Voice Memo	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
AMR MMS	✓	✓	✓	✓				✓		✓	✓	✓	✓
Speech Reco	✓		✓							✓	✓	✓	✓
TTY					✓				✓				
FIR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AEC					✓				✓				
Audio Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TIDE/DCO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

(*): 2nd melody E1 should be started during (and not at beginning of) Melody E1, it corresponds to “game mode” Idem for melody E2.

26.4 Audio Task Compatibilities

The sheets below show the compatibility of all audio tasks with another audio task.

26.4.1 DSP codes <= 33

DSP CODE in B-sample

	Keybeep	Tones	Melody E1	VM play	VM rec	SR enroll	SR update	SR reco	FIR	AEC
Keybeep										
Tones										
Melody E1										
VM play										
VM rec										
SR enroll										
SR update										
SR reco										
FIR										
AEC										

DSP CODE in C-sample

compatibility unknow n??

	Keybeep	Tones	Melody E1	VM play	VM rec	SR enroll	SR update	SR reco	FIR	AEC	Audio Mode	Melody E2
Keybeep												
Tones												
Melody E1												
VM play												
VM rec												
SR enroll												
SR update												
SR reco												
FIR												
AEC												
Audio Mode												
Melody E2												

CAPTION:

can run with this task
can't run with this task.

26.4.2 DSP code 34

feature/feature	0x3416, 0x4180										
	Keybeep	Tones	Melody E1	Voice Memo	AMR MMS	Speech Reco	TTY	FIR	AEC	Audio Mode	TIDE/DCO
Keybeep		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Tones	✓			✓	✓	✓	✓	✓	✓	✓	✓
Melody E1	✓							✓	✓	✓	✓
Voice Memo	✓	✓						✓	✓	✓	✓
AMR MMS	✓	✓						✓	✓	✓	✓
Speech Reco	✓	✓						✓	✓	✓	✓
TTY	✓	✓						✓		✓	✓
FIR	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
AEC	✓	✓	✓	✓	✓	✓		✓		✓	✓
Audio Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
TIDE/DCO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

26.4.3 DSP code 35

feature/feature	0x3507, 0x5190											
	Keybeep	Tones	Melody E1	Melody E2	Voice Memo	AMR MMS	Speech Reco	TTY	FIR	AEC	Audio Mode	TIDE/DCO
Keybeep		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Tones	✓				✓	✓	✓	✓	✓	✓	✓	✓
Melody E1	✓		see (*)						✓	✓	✓	✓
Voice Memo	✓	✓							✓	✓	✓	✓
AMR MMS	✓	✓							✓	✓	✓	✓
Speech Reco	✓	✓							✓	✓	✓	✓
TTY	✓	✓							✓		✓	✓
FIR	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
AEC	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓
Audio Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
TIDE/DCO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

26.4.4 DSP code 36

feature/feature	0x3606, 0x6180											
	Keybeep	Tones	Melody E1	Melody E2	Voice Memo	AMR MMS	Speech Reco	TTY	FIR	AEC	Audio Mode	TIDE/DCO
Keybeep		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Tones	✓				✓	✓	✓	✓	✓	✓	✓	✓
Melody E1	✓		see (*)						✓	✓	✓	✓
Melody E2	✓			see (*)					✓	✓	✓	✓
Voice Memo	✓	✓							✓	✓	✓	✓
AMR MMS	✓	✓							✓	✓	✓	✓
Speech Reco	✓	✓							✓	✓	✓	✓
TTY	✓	✓							✓		✓	✓
FIR	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
AEC	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓
Audio Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
TIDE/DCO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

26.5 Interface Description

26.5.1 Keybeep Generation Functions

26.5.1.1 audio_keybeep_start

```
T_AUDIO_RET audio_keybeep_start( T_AUDIO_KEYBEEP_PARAMETER parameter,
                                 T_RV_RETURN return_path )
```

Description

This function is called to initiate a key beep generation and DTMF generation. The key beep is the generation of two simultaneous sine waves.

Parameters

- **T_AUDIO_KEYBEEP_PARAMETER**

Specifies the characteristic of the keybeep to start.

```
typedef struct {
  UINT16       frequency_beep[2]; // Frequency of the 2 beeps
  INT8         amplitude_beep[2]; // Amplitude of the 2 beeps
  UINT16       duration;
}T_AUDIO_KEYBEEP_PARAMETER;
```

Below the detail of each parameters:

frequency_beep[2] Specifies the frequency of the beeps 1 and 2 in 1 Hz unit. Note the range is [0…2000] Hz. If the frequency value is equal to NO_BEEP, the beep isn't generated.
amplitude_beep[2] Specifies the amplitude of the beeps 1 and 2 in 1 dB unit. Note the range is [-48…0] dB.
duration Specifies the duration of the key beep in ms. Note this duration can't be equal to 20ms.

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	An error is occurred during the execution of this function

Event Return

- **AUDIO_KEYBEEP_STATUS_MSG**

This event is the status send at the end of the keybeep generation or if an error occurred.

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
}T_AUDIO_KEYBEEP_STATUS;
```

The possible values of *status* are:

value	id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped
-1	AUDIO_ERROR	The audio features was not successfully executed

Current restriction of use

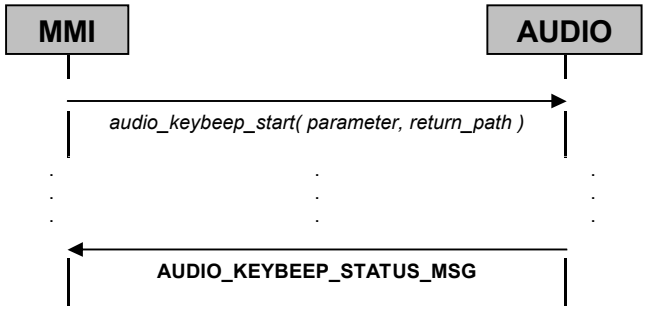
The following restriction of use MUST BE followed by the entity. If it isn't the case, the good functionality of the complete system isn't guarantee.

Note: these following restrictions are only available in the latest version of the software.

- An entity isn't allowed to call this API function if the following audio features is running:
  - A speech recognition task (enroll, update, reco).

Note: this restriction is managed by the Audio entity, therefore the keybeep isn't started if the speech recognition is running.

Process flow



26.5.1.2 audio_keybeep_stop

```
AUDIO_RET audio_keybeep_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the key beep generation.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_keybeep_start*.

Event Return

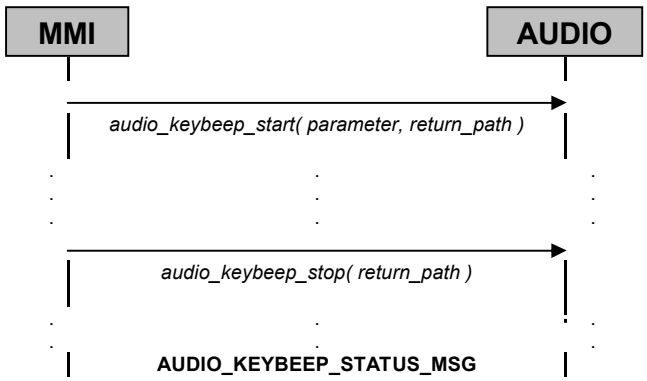
- **AUDIO_KEYBEEP_STATUS_MSG**

C.f. API function *audio_keybeep_start*.

Current restriction of use

none

Process flow



26.5.2 Tones Generation Functions

26.5.2.1 audio_tones_start

```
T_AUDIO_RET audio_tones_start (T_AUDIO_TONES_PARAMETER *p_parameter,
                               T_RV_RETURN return_path)
```

Description

This function is called to initiate the tones generation. The tones are the generation of up to three scheduled sine waves.

Parameters

- **T_AUDIO_TONES_PARAMETER**

Specifies the characteristic of the keybeep to start.

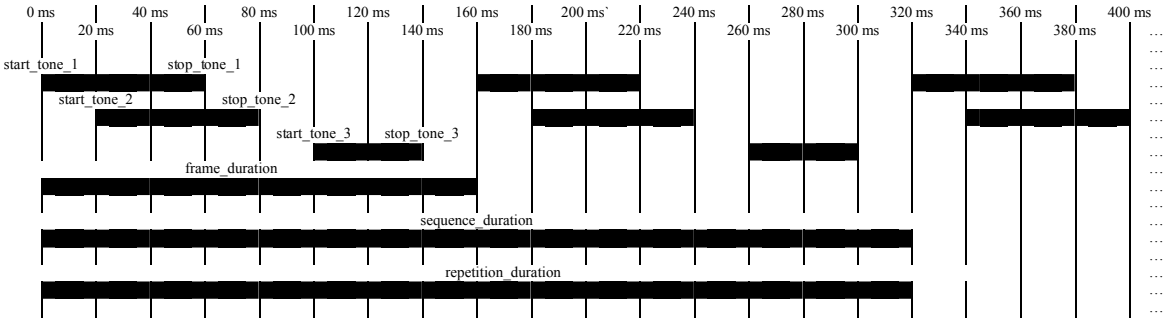
```
typedef struct {
  T_AUDIO_TONE_DESC   tones[3];          // Description of the  3 tones
  UINT16              frame_duration;    // duration of the tones frame
  UINT16              sequence_duration; // duration of the sequence
  UINT16              period_duration;   // duration of the period
  UINT16
}T_AUDIO_TONES_PARAMETER;

typedef struct {
  UINT16   start_tone;    // start date of the tone
  UINT16   stop_tone;     // stop date of the tone
  UINT16   frequency_tone;// frequency of the tone
  INT8     amplitude_tone;// amplitude of the tone
}T_AUDIO_TONE_DESC;
```

Below the detail of each parameters:

start_tone Specifies when the tone 1, 2, 3 must be start in ms unit.
stop_tone Specifies when the tone 1, 2, 3 must be stop in ms unit. **Note the stop_tone > start_tone and stop_tone-start_tone > 20ms.**
frequency_tone Specifies the frequency of the tone 1, 2, 3 in 1 Hz unit. Note the range is [0…2000] Hz. If the frequency value is equal to NO_TONE, the beep isn't generated.
amplitude_tone Specifies the amplitude of the tone 1, 2, 3 in 1 dB unit. Note the range is [-48…0] dB.
frame_duration Specifies the duration of the tones frame (c.f. figure below) in ms unit**. Note this duration can't be equal to 20ms.**
sequence_duration Specifies the duration of the sequence (c.f. figure below) in ms unit**. Note the sequence_duration >= frame_duration.**
period_duration Specifies the duration of the repetition (c.f. figure below) in ms unit**. Note the repetition_duration >= sequence_duration.**
repetition Specifies the number of repetition the tones defined with the parameters above must be played (c.f. figure below). **If the repetition = TONE_INFINITE, the tones is played indefinitely.**

To understand each parameter, please see the figure and example below:



The parameters corresponding to the figure above are:

```
parameter->tone[0].start_tone     = 0;
parameter->tone[0].stop_tone      = 60;
parameter->tone[0].frequency_tone = 520// Hz
parameter->tone[0].amplitude_tone = -24// dB
parameter->tone[1].start_tone     = 20;
parameter->tone[1].stop_tone      = 80;
parameter->tone[1].frequency_tone = 775// Hz
parameter->tone[1].amplitude_tone = -15// dB
parameter->tone[2].start_tone     = 100;
parameter->tone[2].stop_tone      = 140;
parameter->tone[2].frequency_tone = 643 // Hz
parameter->tone[2].amplitude_tone = -6  // dB
parameter->frame_duration         = 160;
parameter->sequence_duration      = 320;
parameter->period_duration        = 320;
parameter->repetition             = TONE_INFINITE;  // infinite tones
```

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_TONES_STATUS_MSG**

This event indicates that the tones task is stopped or an error occured.

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
}T_AUDIO_TONES_STATUS;
```

The possible values of *status* are:

value	id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped
-1	AUDIO_ERROR	The audio features was not successfully executed

Current restriction of use

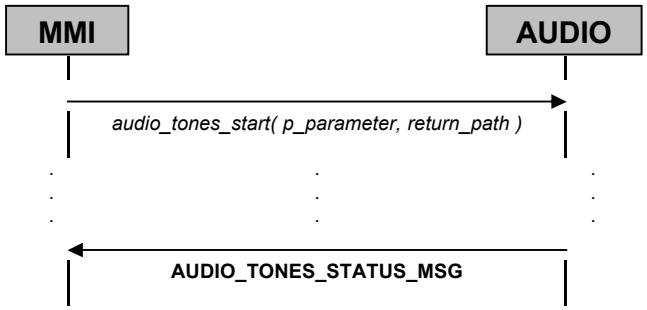
The following restriction of use MUST BE followed by the entity. If it isn't the case, the good functionality of the complete system isn't guarantee.

Note: these following restrictions are only available in the latest version of the software.

- An entity isn't allowed to call this API function if the following audio features is running:
  - A melody E1.
  - A voice memorization (recording).
  - A speech recognition task (enroll, update, reco).

Note: this restriction is managed by the audio entity, therefore the tone isn't started if the speech recognition or voice memorization recording or melody E1 is running.

Process flow



26.5.2.2 audio_tones_stop

```
T_AUDIO_RET audio_tones_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the tones generation**.**

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_keybeep_start*.

Event Return

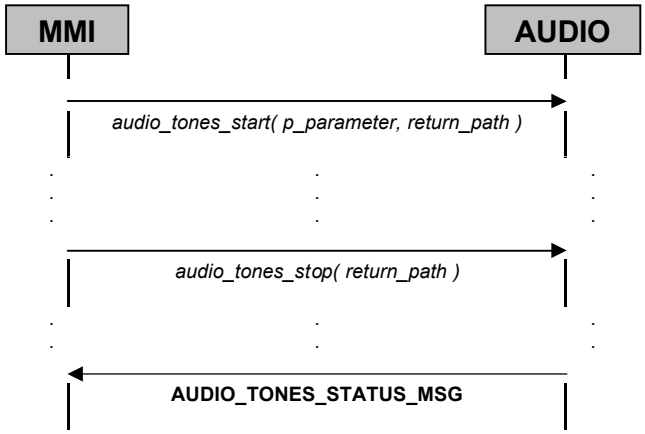
- **AUDIO_TONES_STATUS_MSG**

C.f. API function *audio_tones_start*.

Current restriction of use

C.f. API function *audio_tones_start*.

Process flow



26.5.3 Melody E1 Generation

26.5.3.1 audio_melody_E1_start

```
T_AUDIO_RET audio_melody_E1_start(T_AUDIO_MELODY_E1_PARAMETER *p_parameter,
                                  T_RV_RETURN return_path)
```

Description

This function is called to initiate the melody E1 generation.
Note: two melodies can be run in parallel.

Parameters

- **T_AUDIO_MELODY_E1_PARAMETER**

Specifies the characteristic of the melody to start.

```
typedef struct {
  char   melody_name[AUDIO_MELODY_PATH_NAME_MAX_SIZE];
  // File name of the melody
  BOOL   loopback;    // the melody is played indefinitely
  BOOL   melody_mode; // mode of the melody
}T_AUDIO_MELODY_E1_PARAMETER;
```

Below the detail of each parameters:

melody_name Specifies the file name of the melody. Note that this file name is used by the audio entity to request the melody data to the File Flash System. Moreover, the file name must contain the entire path to access to the melody file. Note the maximum size of the path plus the name is 20 characters.
loopback If *loop_back* = AUDIO_MELODY_LOOPBACK the melody is played indefinitely else one time else if *loop_back* = AUDIO_MELODY_NO_LOOPBACK the melody is played only one time.
melody_mode If *melody_mode* = AUDIO_MELODY_GAME_MODE two melody can be played in parallel in order to use the melody for the game (Note the 8 notes resource is shared between this two melody). If *melody_mode* = AUDIO_MELODY_NORMAL_MODE only one melody is played. So all the 8 notes resource is for this melody.

- **T_RV_RETURN**

C.f. previous section (return mechanism).

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_MELODY_E1_STATUS_MSG**

This event indicates that the melody task is stopped correctly or an error occurred during the execution.

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
}T_AUDIO_MELODY_E1_STATUS;
```

The possible values of *status* are:

value	id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped.
-1	AUDIO_ERROR	The audio features was not successfully executed
-2	AUDIO_MODE_ERROR	A melody is running in normal mode. Therefor, no more melody can not be run.

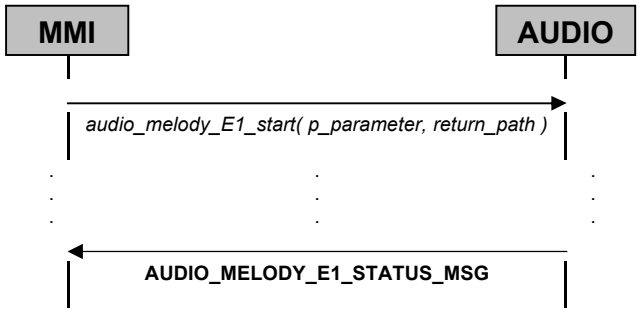
Current restriction of use

The following restriction of use MUST BE followed by the entity. **If it isn't the case, the good functionality of the complete system isn't guarantee.**

Note: these following restrictions are only available in the latest version of the software.

- An entity isn't allowed to call this API function if the following audio features is running:
  - A tone.
  - A voice memorization (recording and playing).
  - A speech recognition (enrollment, update, recognition).
  - Note: this restriction is managed by the audio entity, therefore the melody E1 isn't started if the speech recognition or voice memorization recording or playing and tone is running.
- In normal mode, only one melody can be run.
- Two melodies with the same name can't be run together (i.e. in game mode).

Process flow



26.5.3.2 audio_melody_E1_stop

```
T_AUDIO_RET audio_melody_E1_stop (
                              T_AUDIO_MELODY_E1_STOP_PARAMETER *p_parameter,
                              T_RV_RETURN return_path)
```

Description

This function is called in order to stop the melody generation.

Parameters

- **T_AUDIO_MELODY_E1_STOP_PARAMETER**

Specifies the characteristic of the melody to stop.

```
typedef struct {
  char          melody_name[AUDIO_MELODY_PATH_NAME_MAX_SIZE];
  // File name of the melody
}T_AUDIO_MELODY_E1_PARAMETER;
```

Below the detail of each parameters:

melody_name
Specifies the file name of the melody to stop. Note that this file name must be the name of the melody previously started. Moreover, the file name must contain the entire path to access to the melody file. Note the maximum size of the path plus the name is 20 characters.

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_keybeep_start*.

Event Return

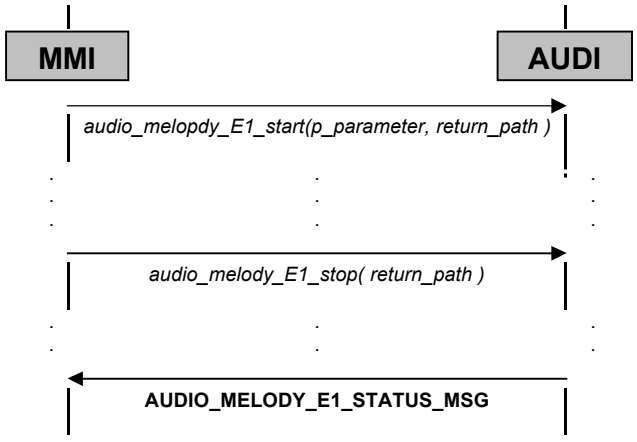
- **AUDIO_MELODY_E1_STATUS_MSG**

C.f. API function *audio_melody_E1_start*.

Current restriction of use

C.f. API function *audio_melody_E1_start*.

Process flow



26.5.4 Voice Memorization Functions

26.5.4.1 audio_vm_record_start

```
T_AUDIO_RET audio_vm_record_start (
            T_AUDIO_VM_RECORD_PARAMETER *p_record_parameter,
            T_AUDIO_TONES_PARAMETER *p_tones_parameter,
            T_RV_RETURN return_path)
```

Description

This function is called to initiate the voice memorization recording phase. Note tones are generated only if the conversation is recording during a call.

Parameters

- **T_AUDIO_VM_RECORD_PARAMETER**

Specifies the parameters using during the voice memorization phase.

```
typedef struct {
  char            memo_name[AUDIO_MEMO_PATH_NAME_MAX_SIZE];
  UINT32          memo_duration;       // maximum duration of the voice memo
  BOOL            compression_mode;    // activate the compression
  UINT16          microphone_gain;     // recording gain applies to microphone
  UINT16          network_gain;        // gain applies to the network voice
}T_AUDIO_VM_RECORD_PARAMETER;
```

Below the detail of each parameters:

memo_name Specifies the file name of the voice memo. Note that this file name is used by the audio entity to request the memo data to the File Flash System. Moreover, the file name must contain the entire path to declare the memo file. Note the maximum size of the path plus the name is 20 characters.
memo_duration Specifies the duration of the voice memo in second unit when the compression of the voice recorded is deactivated. In case of COMPRESSION_MODE, this duration indicates the minimum duration of the voice memo.
microphone_gain Specifies the gain multiplied to the voice sample from the microphone. The format is Q8.8, for example: if microphone_gain = 0x0100, the gain is 1 and if microphone_gain = 0x0080, the gain is 0,5.
network_gain Specifies the gain multiplied to the voice sample from the network (if the mobile is in dedicated mode). The format is Q8.8, for example: if network_gain = 0x0100, the gain is 1 and if network_gain = 0x0080, the gain is 0,5.
compression_mode Activate (COMPRESSION_MODE) or deactivate (NO_COMPRESSION_MODE) the compression of the voice recorded. It means that the silence between two voice activity are compressed.

- **T_AUDIO_TONES_PARAMETER**

See the API function: "*audio_tones_start*". Note that these tones are generated only if the conversation is recording during a call.

- **T_RV_RETURN**

C.f. previous section (return mechanism).

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_VM_RECORD_STATUS_MSG**

This event indicates that the melody task is stopped or an error is occurred.

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
  UINT16     recorded_duration;
}T_AUDIO_VM_RECORD_STATUS;
```

Below the detail of the parameter:

recorded_duration Specifies the size in seconds' unit of the recorded data.

The possible values of *status* are:

value	Id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped.
-1	AUDIO_ERROR	The audio features was not successfully executed

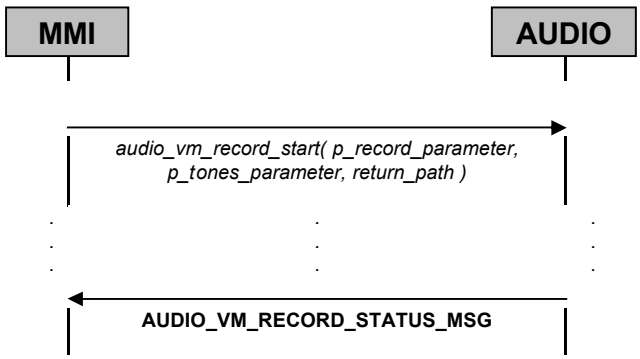
Current restriction of use

The following restriction of use MUST BE followed by the entity. **If it isn't the case, the good functionality of the complete system isn't guarantee.**

Note: these following restrictions are only available in the latest version of the software.

- An entity isn't allowed to call this API function if the following audio features is running:
  - A melody E1.
  - A tone.
  - A speech recognition (enrollment, update, update-check, recognition).
  - A voice memorization playing.
  - Note: this restriction is managed by the audio entity, therefore the Voice memorization recording isn't started if the speech recognition or voice memorization playing or tone or melody E1 is running.
- All directories included in the pathname must be declared before

Process flow



26.5.4.2 audio_vm_record_stop

```
T_AUDIO_RET audio_vm_record_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the current voice memorization record.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

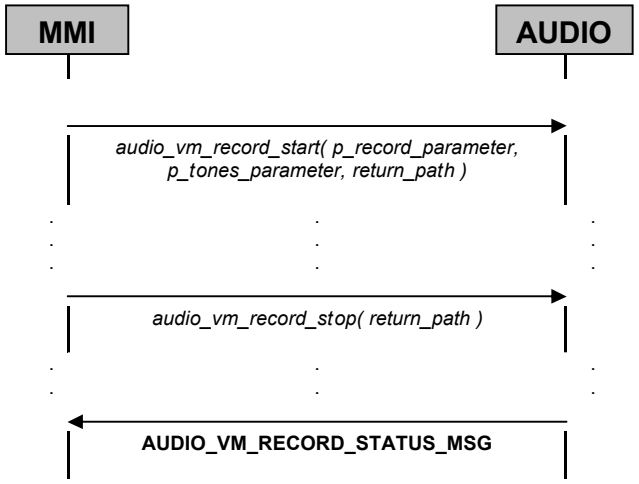
- **AUDIO_VM_RECORD_STATUS_MSG**

C.f. API function *audio_vm_record_start*.

Current restriction of use

C.f. API function *audio_vm_record_start*.

Process flow



26.5.4.3 audio_vm_play_start

```
T_AUDIO_RET audio_vm_play_start (T_AUDIO_VM_PLAY_PARAMETER *p_parameter,
                                 T_RV_RETURN return_path)
```

Description

This function is called to initiate the voice memorization playing phase.

Parameters

- **T_AUDIO_VM_PLAY_PARAMETER**

Specifies the parameters using during the voice memorization phase.

```
typedef struct {
  char           memo_name[AUDIO_MEMO_PATH_NAME_MAX_SIZE];
}T_AUDIO_VM_PLAY_PARAMETER;
```

Below the detail of each parameters:

memo_name
Specifies the file name of the voice memo. Note that this file name is used by the audio entity to request the memo data to the File Flash System. Moreover, the file name must contain the entire path to declare the memo file. Note the maximum size of the path plus the name is 20 characters.

- **T_RV_RETURN**

C.f. previous section (return mechanism).

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_VM_PLAY_STATUS_MSG**

This event indicates that the voice memorization playing task is stopped or an error is occurred.

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
}T_AUDIO_VM_PLAY_STATUS;
```

The possible values of *status* are:

value	id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped.
-1	AUDIO_ERROR	The audio features was not successfully executed

Current restriction of use

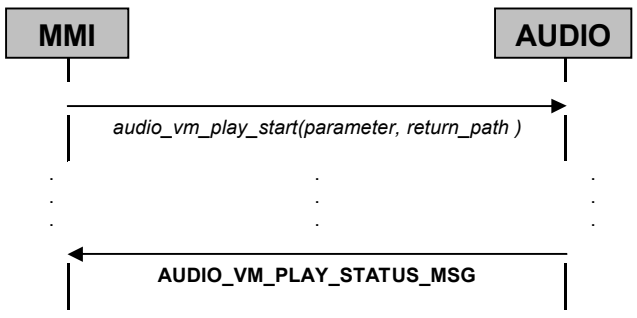
The following restriction of use MUST BE followed by the entity. **If it isn't the case, the good functionality of the complete system isn't guarantee.**

Note: these following restrictions are only available in the latest version of the software.

- An entity isn't allowed to call this API function if the following audio features is running:
  - A melody E1.
  - A speech recognition (enrollment, update, update-check, recognition).
  - A voice memorization recording.

  Note: this restriction is managed by the audio entity, therefore the voice memorization playing isn't started if the speech recognition or voice memorization playing is running.
- All directories included in the pathname must be declared before

Process flow



26.5.4.4 audio_vm_play_stop

```
T_AUDIO_RET audio_vm_play_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the current voice memorization play.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

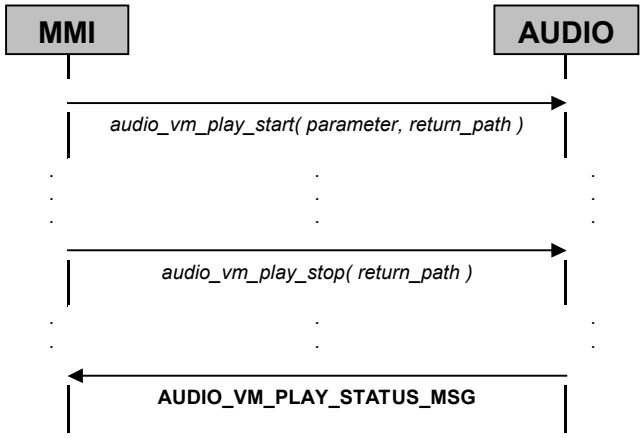
- **AUDIO_VM_PLAY_STATUS_MSG**

C.f. API function *audio_vm_play_start*.

Current restriction of use

C.f. API function *audio_vm_play_start*.

Process flow



26.5.5 MP3 Functions

MP3 controls for MP3 file playing are:

- Start/Stop: start and stop playing MP3 file.
- Pause: pause playing.
- Resume: resume playing **after a pause**

Rules to respect:

The MMI must respect the following rules to play a MP3 file:

- The MMI isn't allowed to play a new MP3 before receiving an Audio MP3 Status.
- The MMI can receive a stop confirmation (Audio MP3 status) in the following cases:
- The MMI requested to stop the playing (audio_mp3_stop function) and the Audio entity confirms this action

- The MP3 decoding is finished and the Audio entity informs the MMI with a stop confirmation message.
- An error occurred from while playing the MP3 file.
- After a pause request (audio_mp3_pause function), the MMI is allowed to request a resume or a stop playing (audio_mp3_resume or audio_mp3_stop).
- The MMI isn't allowed to use the start command during a pause: resume should be used instead.
- Resume commands has no effect outside pause mode.
- The MMI isn't allowed to request two pauses in a row .
- The MMI isn't allowed to request MP3 information before sending a start request (audio_mp3_start function) or after sending a stop request (audio_mp3_stop function).

This section describes how to play a MP3 melody, using the AUDIO SW entity service.

26.5.5.1 audio_mp3_start

```
T_AUDIO_RET audio_mp3_start (  T_AUDIO_MP3_PARAMETER  *p_parameter,
                               T_RV_RETURN             *p_return_path)
```

Description

This function is called to start a MP3 melody generation.

Parameters

- **T_AUDIO_MP3_PARAMETER**

```
typedef struct
{
  char    mp3_name[AUDIO_MP3_PATH_NAME_MAX_SIZE]; // File name of the melody
  BOOL    mono_stereo;                            // channel configuration
  UINT32  size_file_start;                        // size in bytes
} T_AUDIO_MP3_PARAMETER;
```

Below the detail of each parameter:

mp3_name Specifies the file name of the MP3 melody. Note that this file name is used by the audio entity to request the melody data to the File Flash System. Moreover, the file name must contain the entire path to access to the melody file.
Mono_stereo Specifies the configuration of the channel. If *Mono_stereo* = AUDIO_MP3_MONO, the channel configuration is Mono If *Mono_stereo* = AUDIO_MP3_STEREO, the channel configuration is Stereo
size_file_start Specifies the size (in bytes) from where the melody should be started If size_file_start = 0, the melody is played from the beginning of the MP3 file If size_file_start = XXX, the melody is played from the byte XXX.

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return

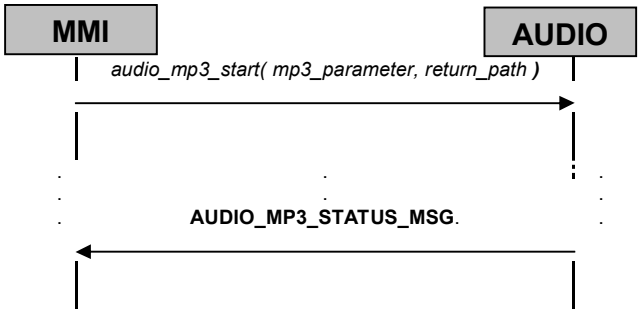
- **AUDIO_MP3_STATUS**

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
} T_AUDIO_MP3_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Process flow



26.5.5.2 audio_mp3_stop

```
T_AUDIO_RET audio_mp3_stop (UINT32 *size_played)
```

Description

This function is called to stop playing a MP3 melody.

Parameters

- **UINT32 *size_played**

This parameter returns the size of the file that has been played before the audio_mp3_stop function was called. This size is in bytes.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

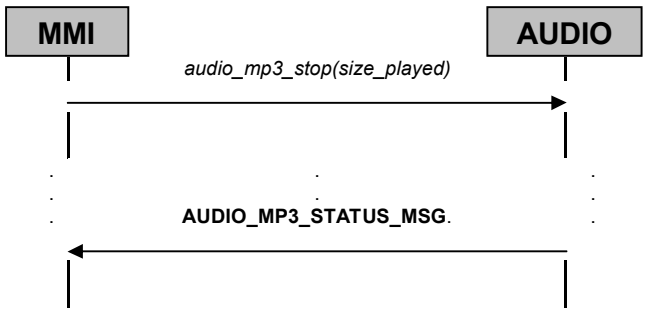
Event Return (if start API was called)

- **AUDIO_MP3_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT16       status;
} T_AUDIO_MP3_STATUS;
```

The possible values of status are:

Value	Id
0	AUDIO_OK
0x0002	C_MP3_SYNC_NOT_FOUND
0x0004	C_MP3_NOT_LAYER3
0x0008	C_MP3_FREE_FORMAT
0x0010	C_MP3_ALG_ERROR
0x0020	C_MP3_DECODING_DELAY
0x04000	C_MP3_CHECK_BUFFER_KO
0x08000	C_MP3_CHECK_BUFFER_DELAY



26.5.5.3 audio_mp3_pause

```
T_AUDIO_RET audio_mp3_pause (void)
```

Description

This function is called to pause playing a MP3 melody. The MP3 melody can be restarted using the audio_mp3_resume function.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return (if start API was called)

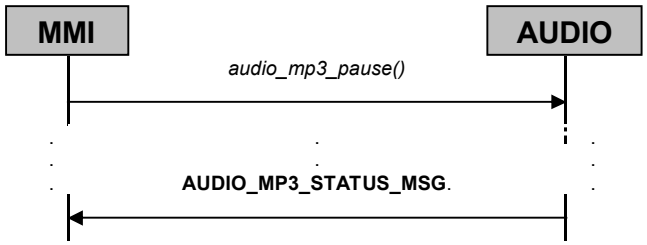
- **AUDIO_MP3_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
} T_AUDIO_MP3_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Process flow



26.5.5.4 audio_mp3_resume

```
T_AUDIO_RET audio_mp3_resume (void)
```

Description

This function is called to resume a MP3 melody, after a pause.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return (if start API was called)

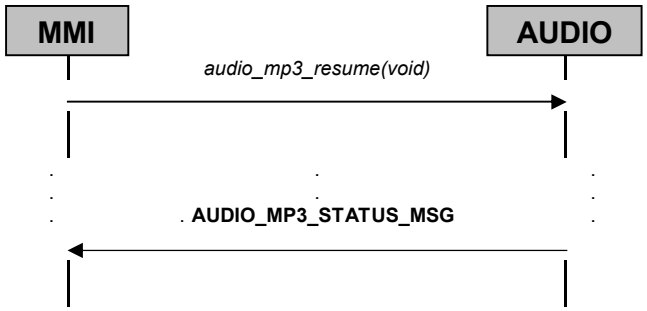
- **AUDIO_MP3_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
} T_AUDIO_MP3_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Process flow



26.5.5.5 audio_mp3_info

```
T_AUDIO_RET audio_mp3_info (void)
```

Description

This function is called to request information about the currently decoded MP3 frame.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return (if start API was called)

- **AUDIO_MP3_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
  T_MP3_HEADER_INFO info;
} T_AUDIO_MP3_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

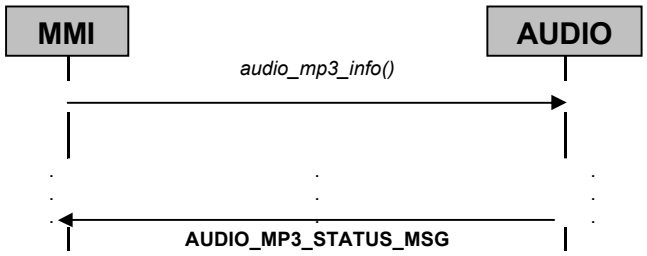
info (T_MP3_HEADER_INFO)

Structure containing information about current MP3 frame:

Field	Type	Possible values
frequency	UINT8	C_MP3_HEADER_FREQ_48000 C_MP3_HEADER_FREQ_44100 C_MP3_HEADER_FREQ_32000 C_MP3_HEADER_FREQ_24000 C_MP3_HEADER_FREQ_22050 C_MP3_HEADER_FREQ_16000 C_MP3_HEADER_FREQ_12000 C_MP3_HEADER_FREQ_11025 C_MP3_HEADER_FREQ_8000
bitrate	UINT8	C_MP3_HEADER_BITRATE_320 C_MP3_HEADER_BITRATE_256 C_MP3_HEADER_BITRATE_224 C_MP3_HEADER_BITRATE_192 C_MP3_HEADER_BITRATE_160 C_MP3_HEADER_BITRATE_128 C_MP3_HEADER_BITRATE_112 C_MP3_HEADER_BITRATE_96 C_MP3_HEADER_BITRATE_80 C_MP3_HEADER_BITRATE_64 C_MP3_HEADER_BITRATE_56 C_MP3_HEADER_BITRATE_48 C_MP3_HEADER_BITRATE_40 C_MP3_HEADER_BITRATE_32
mpeg_id	UINT8	C_MP3_HEADER_MPEGID_1 C_MP3_HEADER_MPEGID_2 C_MP3_HEADER_MPEGID_2_5

layer	UINT8	C_MP3_HEADER_LAYER_I C_MP3_HEADER_LAYER_II C_MP3_HEADER_LAYER_III Note: different layer values than LAYER III can be returned in theory. But this is a LAYER III only decoder.
padding	BOOL	TRUE, FALSE
private	UINT8	0, 1
channel	UINT8	C_MP3_HEADER_STEREO C_MP3_HEADER_JSTEREO C_MP3_HEADER_DUAL_MONO C_MP3_HEADER_MONO
copyright	BOOL	TRUE, FALSE
original	BOOL	TRUE, FALSE
emphasis	UINT8	C_MP3_HEADER_EMPHASIS_NONE C_MP3_HEADER_EMPHASIS_50_15 C_MP3_HEADER_EMPHASIS_CCIT_J17

Process flow



26.5.6 How to use the MP3 APIs

26.5.6.1 "Pause" MP3 in order to play another melody

If the user wants to "pause" the MP3 melody in order to play an other melody (for example Midi ringer), it is necessary to stop the MP3 melody and then restart it. Indeed, due to Hardware constraints the MP3 can't be in "pause" mode when an other melody is playing.

Example of code – the user is listening to a MP3 melody when a Midi ringer needs to be played:

- Start the MP3 melody from the beginning:

```
strcpy(mp3_parameter.mp3_name,"/mp3/mp3_file");
mp3_parameter.mono_stereo = AUDIO_MP3_MONO;
mp3_parameter.size_file_start = 0;

if (audio_mp3_start(&mp3_parameter, return_path) == AUDIO_ERROR)
 {
        *error_type = FUNCTION_ERROR;
        return (audio_test_regr_return_verdict(*error_type));
}
```

- Midi Ringer must be played
  - First Stop the MP3 melody:

```
if (audio_mp3_stop(size_played) == AUDIO_ERROR)
{
        *error_type = MEMORY_ERROR;
         return (audio_test_regr_return_verdict(*error_type));
 }
```

- Play the Midi ringer
Use the start function to resume the MP3 melody

```
mp3_parameter.size_file_start = *size_played;

 if (audio_mp3_start(&mp3_parameter, return_path) == AUDIO_ERROR)
 {
    *error_type = FUNCTION_ERROR;
    return (audio_test_regr_return_verdict(*error_type));
}
```

26.5.6.2 "Pause" MP3 and resume it

If the user want to pause the MP3 melody and no melody is played in parallel:

- Start the MP3 melody from the beginning:

```
 strcpy(mp3_parameter.mp3_name,"/mp3/mp3_file");
mp3_parameter.mono_stereo = AUDIO_MP3_MONO;
mp3_parameter.size_file_start = 0;

if (audio_mp3_start(&mp3_parameter, return_path) == AUDIO_ERROR)
 {
    *error_type = FUNCTION_ERROR;
    return (audio_test_regr_return_verdict(*error_type));
 }
```

- Pause the MP3 melody:

```
if (audio_mp3_pause() == AUDIO_ERROR)
{
  *error_type = FUNCTION_ERROR;
  return (audio_test_regr_return_verdict(*error_type));
}
```

- And then resume the MP3 melody thanks to the resume function:

```
if (audio_mp3_resume() == AUDIO_ERROR)
{
   *error_type = FUNCTION_ERROR;
   return (audio_test_regr_return_verdict(*error_type));
}
```

26.6 Audio Mode Configuration

26.6.1 Introduction

This section sums up all the API functions useful to handle all possible the audio paths embedded in a mobile. These API functions can be grouped in several family of use:

- **The MMI family:**
  These API functions are dedicated to facilitate the creation, the calibration, and the change of the audio mode.
  Note: An audio mode is a fixed setting of all audio features embedded in the mobile. For example, during an incoming call, the mobile rings so this task uses a particular setting of all audio features therefore it corresponds to a particular audio mode. In this case, the audio mode is called RING_MODE.
- **The full access family:**
  These API functions are dedicated to permit a direct tuning of all audio module involved in the audio paths. For example, with these functions, you can directly tune the PGA gain of the microphone connected to the Analog Base Band.

26.6.2 The MMI Family

This section describes all the API functions belong to the MMI family. Before to define these API functions, the first step is to define several vocabulary and concept used in this chapter.

- Audio mode:
  An audio mode is a fixed setting of all audio features embedded in the mobile. For example, during an incoming call, the mobile rings so this task uses a particular setting of all audio features therefore it corresponds to a particular audio mode. In this case, the audio mode is called RING_MODE.
  The list of all standard audio modes is listed below, but there's a possibility to extend this list.
  Moreover, the audio setting of an audio mode can be grouped in several family of audio setting:
- Audio path setting:
  This group of setting is used to define the audio path used. The different audio paths are:
  - **GSM normal path**: voice samples are exchanged between GSM network and GSM Analog Base Band.
  - **Bluetooth Cordless path**: voice samples are exchanged between the GSM Analog Base Band and the Bluetooth module.
  - **Bluetooth Headset path**: voice samples are exchanged between GSM network and Bluetooth module.
  - **DAI encoder path**: path to test the speech encoder and its DTX functions.
  - **DAI decoder path**: path to test the speech decoder and its DTX functions.
  - **DAI acoustic path**: path to test the acoustic devices and the audio A/D and D/A devices.
- Microphone voice path setting:
  This group of setting configures the audio voice path of the microphone.
- Speaker voice path setting:
  This group of setting configures the audio voice path of the speaker.
- Microphone speaker loop setting:
  This group of setting configures the audio module involved in the microphone and speaker voice loop.
- Speaker audio stereo path:
  This group of setting configures the audio stereo path of the speaker.

26.6.2.1 Audio mode file structure

The audio mode is described by the structure below *T_AUDIO_MODE*. So for each audio mode (i.e. game, audio off, ringer, handheld…), a *T_AUDIO_MODE* variable is saved in a flash file in the folder */aud/* with the extension .cfg. The file name is specified by the customer (c.f. audio_mode_save/load function)

26.6.2.1.1 Analog Base Band – TRITON

26.6.2.1.1.1 T_AUDIO_MODE

Specifies the structure of each audio mode:

```
typedef struct
      {
        /* group of setting to define the audio path used */
        T_AUDIO_VOICE_PATH_SETTING              audio_path_setting;
        /* group of setting to configure the audio voice path of the microphone */
        T_AUDIO_MICROPHONE_SETTING              audio_microphone_setting;
        /* group of setting to configure the audio voice path of the speaker */
        T_AUDIO_SPEAKER_SETTING                 audio_speaker_setting;
        /* group of setting to configure the audio stereo path of the speaker */
        T_AUDIO_STEREO_SPEAKER_SETTING          audio_stereo_speaker_setting;
        /* group of setting to configure the audio mode involved */
        /* in the microphone and speaker loop */
        T_AUDIO_MICROPHONE_SPEAKER_LOOP_SETTING audio_microphone_speaker_loop_setting;
        /* group of settings to configure audio features common to */
        /* microphone and speaker */
        T_AUDIO_MICROPHONE_SPEAKER_SETTING audio_microphone_speaker_setting;
      }
      T_AUDIO_MODE;
```

26.6.2.1.1.2 T_AUDIO_VOICE_PATH_SETTING

This parameter specifies the audio path mode.

```
      /* audio path used */
      typedef UINT8 T_AUDIO_VOICE_PATH_SETTING;
```

The different values are:

Value	Path
AUDIO_GSM_VOICE_PATH	GSM normal
AUDIO_BLUETOOTH_CORDLESS_VOICE_PATH	Bluetooth cordless
AUDIO_BLUETOOTH_HEADSET_PATH	Bluetooth headset
AUDIO_DAI_ENCODER	DAI encoder
AUDIO_DAI_DECODER	DAI decoder
AUDIO_DAI_ACOUSTIC	DAI acoustic

26.6.2.1.1.3 T_AUDIO_MICROPHONE_SETTING

Specifies the setting of the microphone voice path,

```
          typedef struct
          {
            /* mode of the microphone */
            INT8                      mode;
            /* Setting of the current mode */
            T_AUDIO_MICROPHONE_MODE  setting;
          }
          T_AUDIO_MICROPHONE_SETTING;
```

Where the microphone modes are :

```
    typedef union
```

```
      {
        /* handheld mode parameters */
        T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB      handset_25_6db;
       /* handheld mode parameters */
        T_AUDIO_MICROPHONE_MODE_HEADSET_4_9_DB      headset_4_9db;
       /* handheld mode parameters */
        T_AUDIO_MICROPHONE_MODE_HEADSET_25_6DB      headset_25_6db;
       /* handheld mode parameters */
        T_AUDIO_MICROPHONE_MODE_HEADSET_18DB        headset_18db;
       /* Aux mode parameters */
        T_AUDIO_MICROPHONE_MODE_AUX_4_9DB           aux_4_9db;
       /* Aux mode parameters */
        T_AUDIO_MICROPHONE_MODE_AUX_28_2DB          aux_28_2db;
              /* handfree mode parameters */
        T_AUDIO_MICROPHONE_MODE_FM_MONO             fm_mono;
        /* headset mode parameters */
        T_AUDIO_MICROPHONE_MODE_CARKIT              carkit;
        /* FM */
        T_AUDIO_MICROPHONE_MODE_FM                  fm;
      }
      T_AUDIO_MICROPHONE_MODE;

typedef struct
      {
        /* gain of the microphone */
        INT8    gain;
        /* microphone output bias */
        INT8    output_bias;
        /* coefficients of the microphone FIR */
        T_AUDIO_FIR_COEF    fir;
         /* ANR configuration */
         T_AUDIO_ANR_CFG     anr;
         /* ES configuration */
         T_AUDIO_ES_CFG      es;
      }
      T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB;

typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_HEADSET_4_9_DB;
typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_HEADSET_25_6DB;
typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_HEADSET_18DB;
typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_AUX_4_9DB;
typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_AUX_28_2DB;
typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_CARKIT;

typedef struct
      {
        /* gain of the microphone */
        INT8    gain;
        /* microphone output bias */
        INT8    output_bias;
                    /* microphone output bias */
        INT8    extra_gain;
      }
      T_AUDIO_MICROPHONE_MODE_FM_MONO;

typedef T_AUDIO_MICROPHONE_MODE_FM_MONO T_AUDIO_MICROPHONE_MODE_FM;
```

Mode Specifies the mode of the microphone: **AUDIO_MICROPHONE_MODE_HANDSET_25_6DB** or **AUDIO_MICROPHONE_MODE_HEADSET_4_9_DB** or **AUDIO_MICROPHONE_MODE_HEADSET_25_6DB** or **AUDIO_MICROPHONE_MODE_HEADSET_18DB or AUDIO_MICROPHONE_MODE_AUX_4_9DB or AUDIO_MICROPHONE_MODE_AUX_28_2DB or AUDIO_MICROPHONE_MODE_CARKIT or T_AUDIO_MICROPHONE_MODE_FM_MONO or T_AUDIO_MICROPHONE_MODE_FM**	
All the modes are available in GSM, bluetooth cordless voice and all DAI path mode.	
	Gain gain of the microphone in 1dB unit. The range is from –12 dB to 12 dB. Note if the gain is equal to AUDIO_MICROPHONE_MUTE, the microphone is muted.

	output_bias Specifies the output voltage of the microphone. This value could be 2.0V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_0V) or 2.5 V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_5V).
	extra_gain **Specifies the FM gain. The range is -2dB to 14dB in 2dB steps which corresponds to 0 to 8. Applicable only for `T_AUDIO_MICROPHONE_MODE_FM_MONO and T_AUDIO_MICROPHONE_MODE_FM`.**
	fir_coef List of the 31 coefficients of the FIR of the microphone. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000. **Note: the FIR is available only in DAI and GSM path mode.**
	anr *See 28.6.2.1.1.3.1 for details.*
	es *See 28.6.2.1.1.3.2 for details*

26.6.2.1.1.3.1 T_AUDIO_ANR_CFG

ANR (Ambiant Noise Reduction) module allows reducing the noise present in the speech uplink path.

ANR is only available in TCS 3.x software except TCS 3.0. Other software versions do not include this structure in the microphone settings.
ANR module only works in DAI acoustic and GSM path mode.

ANR settings are inside following structure of the microphone settings:

```
typedef struct
{
  BOOLEAN  anr_enable;
  INT16    min_gain;
  INT8     div_factor_shift;
  UINT8    ns_level;
}
T_AUDIO_ANR_CFG;
```

ANR excepted noise attenuation (dB) = Temp. Att. (dB) + Spec. Att. (dB).

anr_enable Enable/disable the ANR (Ambiant Noise Reduction) module: 0- disable 1- enable
In case of Read Access, the following parameters are valid only if anr_enable = 1.
min_gain Temp. Att. (dB): temporal attenuation applied on signal detected as noise, considering speech isn't attenuated. Format is Q15. Temp. Att. (dB) = 20*log(d_anr_min_gain/32768); Ex: d_anr_min_gain = 0x4000 -> Temp. Att. = -6dB **Recommended value is 0x3313 (-8 dB)**
div_factor_shift Used to control variations of temporal attenuation. Time periods where signal is considered as noise are attenuated. In order to avoid erroneous speech attenuation, this value permits to adjust the freezing of the gain after the speech detection. **Recommended value is -2.**

anr_ns_level
Spec. Att. (dB) : spectral subtraction in 6dB steps.
Ex: d_anr_ns_level | Spec. Att. (dB)

Ex: -1 | 0 dB (*)
0 | -x dB (**)
1 | -6 dB
2 | -12 dB
(*) Customers shouldn't use ANR2.0 without spectral subtraction ;
(**) ANR 2.0 performs the maximum of spectral subtraction depending on the incoming signal.
Recommended value is 1 (-6 dB).

26.6.2.1.1.3.2 T_AUDIO_ES_CFG

The echo suppressor (ES) role is to control the residual echo in a speakerphone application, where the AEC is unable to cancel the entire echo in the uplink due to non-ideal acoustical environment such as a non-linear loudspeaker response for example. Please refer to [8] for an overview of the module.

ES is only available in TCS 3.x software except TCS 3.0. Other software versions do not include this structure in the microphone settings.
ES module only works in DAI acoustic and GSM path mode.

ES settings are inside following structure of the microphone settings:

```
typedef struct
{
  BOOLEAN es_enable;
  UINT8   es_behavior;
  UINT8   es_mode;
  INT16   es_gain_dl;
  INT16   es_gain_ul_1;
  INT16   es_gain_ul_2;
  INT16   tcl_fe_ls_thr;
  INT16   tcl_dt_ls_thr;
  INT16   tcl_fe_ns_thr;
  INT16   tcl_dt_ns_thr;
  INT16   tcl_ne_thr;
  INT16   ref_ls_pwr;
  UINT16  switching_time;
  UINT16  switching_time_dt;
  UINT16  hang_time;
  INT16   gain_lin_dl_vect[4];
  INT16   gain_lin_ul_vect[4];
}
T_AUDIO_ES_CFG;
```

es_enable
Enable/disable the echo suppressor module:
0- disable
1- enable

In case of Read Access, the following parameters are valid only if es_enable = 1.

es_behavior
Permit to setup pre-defined ES behavior as described in [9]:
0- Behavior 1
1- Behavior 1a
2- Behavior 2a
3- Behavior 2b
4- Behavior 2c
5- Behavior 2c_idle
6- Custom: all parameters setup by the user

In 'custom' mode, following parameters must be set. Custom mode isn't recommended.
In other modes, following parameters aren't used.

es_mode
Bitmap defining the ES mode:

bit		
0	ES UL	0- Disable ES on UL path 1- Enable ES on UL path
1	ES DL	0- Disable ES on DL path 1- Enable ES on DL path
2	CNG	0- Disable CNG* algorithm 1- Enable CNG* algorithm
3	NSF	0- Disable NSF** algorithm 1- Enable NSF** algorithm
4	ALS UL	0- Disable ALS*** on UL path 1- Enable ALS*** on UL path
5	ALS DL	0- Disable ALS*** on DL path 1- Enable ALS*** on DL path

* CNG = Comfort Noise Generation
** NSF=Noise Floor
*** ALS = Attenuation Level Smoothing

Notes:
- **Disabling ES UL has no sense**
- **CNG and NSF mustn't be enabled together**

es_gain_dl
es_gain_dl is the receive loss compensation.

es_gain_ul_1
es_gain_ul_1 is the coupling loss compensation.

es_gain_ul_2
es_gain_ul_2 is the near-end propagation loss compensation.

tcl_fe_ls_thr
d_tcl_fe_ls_thr is the TCL reference threshold in far-end mode for loud signals. This value is in Q15 format.

tcl_dt_ls_thr
d_tcl_fd_ls_thr is the TCL reference threshold in double-talk mode for loud signals. This value is in Q15 format

tcl_fe_ns_thr
d_tcl_fe_ns_thr is the TCL reference threshold in far-end mode for nominal signals. This value is in Q15 format

tcl_dt_ns_thr
d_tcl_fd_ns_thr is the TCL reference threshold in double-talk mode for nominal signals. This value is in Q15 format

tcl_ne_thr
d_tcl_ne_thr is the TCL reference threshold in near-end mode. This value is in Q15 format

ref_ls_pwr
d_ref_ls_pwr is the TCL reference threshold in near-end mode. This value is in Q15 format

switching_time *d_ switching_time_dt* is the switching time value in milliseconds.
switching_time_dt *d_ switching_time_dt* is the double-talk switching time value in milliseconds.
hang_time *d_hang_time* is the hangover time for switching
gain_lin_dl_vect Table containing downlink linear attenuation levels per state: gain_lin_dl_vect[0] - idle state gain_lin_dl_vect[1] - double talk gain_lin_dl_vect[2] - far-end gain_lin_dl_vect[3] - near-end Format is Q15.
gain_lin_ul_vect Table containing uplink linear attenuation levels per state: gain_lin_ul_vect[0] - idle state gain_lin_ul_vect[1] - double talk gain_lin_ul_vect[2] - far-end gain_lin_ul_vect[3] - near-end Format is Q15.

26.6.2.1.1.4 T_AUDIO_SPEAKER_SETTING

Specifies the characteristic of the speaker voice path.

```
typedef struct
      {
        /* mode of the speaker */
        INT8                    mode;
        /* Setting of the current mode */
        T_AUDIO_SPEAKER_MODE    setting;
      }
      T_AUDIO_SPEAKER_SETTING;
```

where the speaker modes are:

```
typedef union
    {
     /* handheld mode parameters */
     T_AUDIO_SPEAKER_MODE_HANDHELD handheld;
     /* handfree mode parameters */
     T_AUDIO_SPEAKER_MODE_HANDFREE handfree;
     /* headset mode parameters */
     T_AUDIO_SPEAKER_MODE_HEADSET  headset;

     T_AUDIO_SPEAKER_MODE_AUX          aux;
     T_AUDIO_SPEAKER_MODE_CARKIT carkit;
    }
    T_AUDIO_SPEAKER_MODE;
```

Where the speaker modes are,

```
typedef struct
    {
```

```
    /* gain of the speaker */
    INT8    gain;
    /* use the audio filter */
    INT8  audio_filter;
     /* use the audio highpass filter */
      INT8  audio_highpass_filter;
    /* extra gain of the speaker */
    INT8   extra_gain;
      /* coefficients of the speaker FIR */
      T_AUDIO_FIR_COEF    fir;
      /* Limiter parameter */
      T_AUDIO_LIMITER_CFG   limiter;
      /* IIR filter parameters */
      T_AUDIO_IIR_CFG      iir;
  }
  T_AUDIO_SPEAKER_MODE_HANDHELD;

typedef struct
  {
    /* gain of the speaker */
    INT8    gain;
    /* use the audio filter */
    INT8  audio_filter;
     /* use the audio highpass filter */
      INT8  audio_highpass_filter;
    /* extra gain of the speaker */
    INT8   extra_gain;
      /* coefficients of the speaker FIR */
      T_AUDIO_FIR_COEF    fir;
      /* Limiter parameter */
      T_AUDIO_LIMITER_CFG   limiter;
      /* IIR filter parameters */
      T_AUDIO_IIR_CFG      iir;
  }
  T_AUDIO_SPEAKER_MODE_HANDFREE;

typedef struct
  {
    /* gain of the speaker */
    INT8    gain;
    /* use the audio filter */
    INT8  audio_filter;
     /* use the audio highpass filter */
      INT8  audio_highpass_filter;
      /* coefficients of the speaker FIR */
      T_AUDIO_FIR_COEF    fir;
      /* Limiter parameter */
      T_AUDIO_LIMITER_CFG   limiter;
      /* IIR filter parameters */
      T_AUDIO_IIR_CFG      iir;
  }
  T_AUDIO_SPEAKER_MODE_HEADSET;

typedef struct
  {
    /* gain of the speaker */
    INT8    gain;
    /* use the audio filter */
    INT8  audio_filter;
```

```
        /* use the audio highpass filter */
         INT8  audio_highpass_filter;
         /* coefficients of the speaker FIR */
         T_AUDIO_FIR_COEF    fir;
         /* Limiter parameter */
         T_AUDIO_LIMITER_CFG   limiter;
         /* IIR filter parameters */
         T_AUDIO_IIR_CFG      iir;
    }
    T_AUDIO_SPEAKER_MODE_AUX;

typedef struct
    {
      /* gain of the speaker */
      INT8    gain;
      /* use the audio filter */
      INT8  audio_filter;
       /* use the audio highpass filter */
        INT8  audio_highpass_filter;
        /* coefficients of the speaker FIR */
        T_AUDIO_FIR_COEF    fir;
        /* Limiter parameter */
        T_AUDIO_LIMITER_CFG   limiter;
        /* IIR filter parameters */
        T_AUDIO_IIR_CFG      iir;
    }
    T_AUDIO_SPEAKER_MODE_CARKIT;
```

Mode
Specifies the mode of the microphone: **AUDIO_SPEAKER_MODE_HANDHELD**, AUDIO_SPEAKER_MODE_HANDFREE or AUDIO_SPEAKER_MODE_HEADSET or AUDIO_SPEAKER_MODE_AUX or AUDIO_SPEAKER_MODE_CARKIT.

All these modes are available in GSM, bluetooth cordless voice and all DAI path mode.

gain
Specifies the gain in 1 dB unit of the speaker. The range is from –6 dB to 6 dB.

audio_filter
Add an audio filter in the speaker path in order to enhance the audio quality. The filter is added if *audio_filter* = AUDIO_SPEAKER_FILTER_ON else the filter is bypassed if *audio_filter* = AUDIO_SPEAKER_FILTER_OFF. The frequency response of this hardware filter is the following:

Frequency Response	Gain relative to reference gain at 1kHz	Min	Typ	Max	Units
<= 100 Hz				-20	DB
100 Hz to 200 Hz				-10	dB
300 Hz to 400 Hz		-2	0	+1	dB
400 Hz to 3300 Hz		-1	0	+1	dB
3300 Hz to 3400 Hz		-2	0	+1	dB
4000 Hz to 4600 Hz				-17	dB
4600 Hz to 6000 Hz				-40	dB
>= 6000 Hz				-45	dB

WARNING: IF THE FILTER IS BYPASSED, THE GAIN IS EQUAL TO 0 AND THE VOLUME TOO (c.f. speaker volume API function).

audio_highpass_filter
Add or bypass the high-pass part of the audio filter: AUDIO_SPEAKER_HIGHPASS_FILTER_ON to add it, AUDIO_SPEAKER_HIGHPASS_FILTER_OFF to bypass it.

	extra_gain **Extra gain for AUDIO_SPEAKER_HANDHELD and AUDIO_SPEAKER_HANDFREE modes.** **For AUDIO_SPEAKER_HANDHELD the extra gain values are AUDIO_SPEAKER_SPK_GAIN_8_5DB, AUDIO_SPEAKER_SPK_GAIN_2_5DB, AUDIO_SPEAKER_SPK_GAIN_MINUS_3_5DB and AUDIO_SPEAKER_SPK_GAIN_MINUS_22_5DB.** **For AUDIO_SPEAKER_HANDFREE the extra gain values are AUDIO_EAR_GAIN_MINUS_11DB and AUDIO_EAR_GAIN_1DB.**
	fir_coef List of the coefficient of the FIR of the speaker. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000. **Note: the FIR is available only in DAI and GSM path mode.** **The FIR filter is replaced by IIR filter in TCS3.x software, except TCS 3.0.**
	limiter *See 28.6.2.1.1.4.1 for details.*
	iir *See 28.6.2.1.1.4.2 for details.*

26.6.2.1.1.4.1 T_AUDIO_LIMITER_CFG

Limiter aim is to avoid using the non-linear regions of the speaker response in order to avoid audio saturation/distortion. Please refer to [5] for an overview of the module,

Limiter is only available in TCS 3.x software except TCS 3.0. Other software versions do not include this structure in the speaker settings.
Limiter module only works in DAI acoustic and GSM path mode.

Limiter settings are inside following structure of the speaker settings:

```
typedef struct
{
  BOOLEAN  limiter_enable;
  UINT16   block_size;
  UINT16   slope_update_period;
  UINT16   nb_fir_coefs;
  INT16    filter_coefs[16];
  UINT16   thr_low_0;
  INT16    thr_low_slope;
  UINT16   thr_high_0;
  INT16    thr_high_slope;
  INT16    gain_fall;
  INT16    gain_rise;
}
T_AUDIO_LIMITER_CFG;
```

limiter_enable Enable/disable the limiter 0- disable 1- enable
In case of Read Access, the following parameters are valid only if limiter_enable = 1.
block_size Number of samples in an input block. **Currently, mandatory value is 160.**
slope_update_period Number of samples between each update of the limiter slope. It must be a divider of block_size. **Recommended value is 160.**
nb_fir_coefs Number of coefficients in the filter. It must be an odd number. Maximum number is 31. **Recommended value is 31.**

filter_coefs
Array containing the filter coefficients. This array must contains (nb_fir_coefs-1)/2+1 coefficients. The filter being a symmetric one, other coefficients do not need to be saved in the array.

thr_low_0 / thr_low_slope
thr_high_0 / thr_high_slope

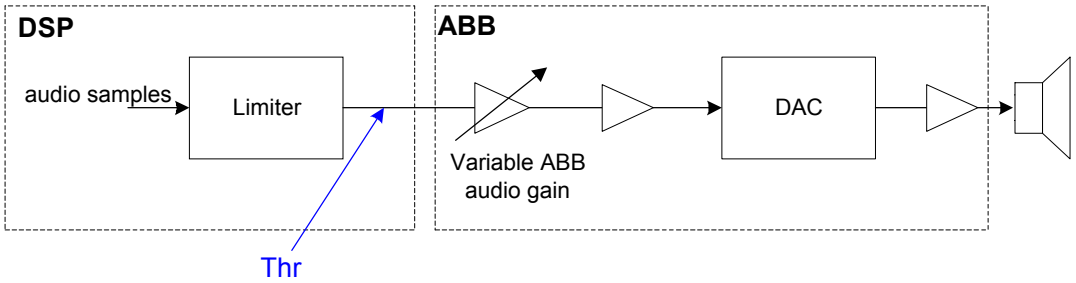
thr_X_0 (range 0..32767)
percentage of the maximum level of signal wanted at the output of the limiter with respect to the maximum possible level set to 1. It has to be multiplied by 32767 to be expressed with only 1 bit significant for integer part and 15 for decimal part.

thr_X_slope (range -30..+6)
Slope threshold above which signal has to be decreased. It is expressed in dB.
Minimum and maximum values depend on possible values for voiceband downlink control register.

These values permit to define Thr(low) and Thr(high) characteristic function to volume setup in the ABB.
Thr(low) and Thr(high) define the maximum level of the signal wanted at the output of the **limiter**:

- Thr(low) for the low frequency part of the signal
- Thr(high) for the high frequency part of the signal
-

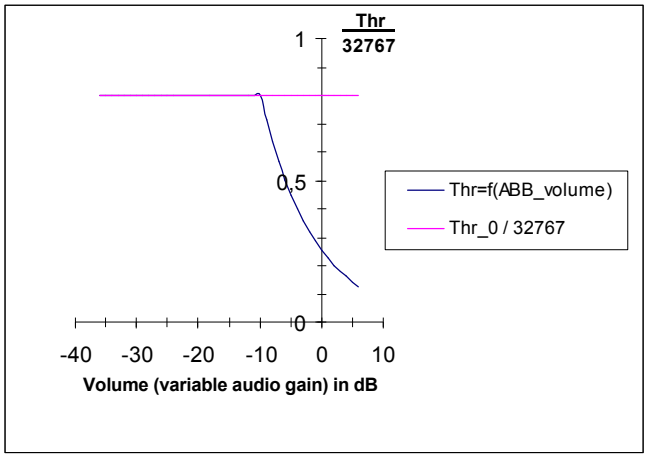
: Following scheme shows a model of the audio patch after the limiter:



Thr(low) or Thr(high) are processed using the following characteristic:

- For $Volume_{(dB)} <= thr\_low\_slope_{(dB)}$,
  $Thr(low) = thr\_low\_0$
- For $Volume_{(dB)} > thr\_low\_slope_{(dB)}$,
  $Thr(low) = thr\_low\_0 \times thr\_low\_slope_{(lin)} / Volume_{(lin)}$
  or
  $Thr(low) = thr\_low\_0 \times 10^{(thr\_low\_slope(dB) - Volume(dB))/20}$

- For $Volume_{(dB)} <= thr\_high\_slope_{(dB)}$,
  $Thr(high) = thr\_high\_0$
- For $Volume_{(dB)} > thr\_high\_slope_{(dB)}$,
  $Thr(high) = thr\_high\_0 \times thr\_high\_slope_{(lin)} / Volume_{(lin)}$
  or
  $Thr(high) = thr\_high\_0 \times 10^{(thr\_high\_slope(dB) - Volume(dB))/20}$



NB:

The name of the thr_low_slope on the L1 code is thr_ABB_vol_low.
The name of the thr_high_slope on the L1 code is thr_ABB_vol_high.

gain_fall
Decrease the slope when saturation has been detected on the previous block of samples. Format is Q15.
Recommended value is 26214
Ex: gain_fall = 26214 => slope(n+1) = 0.8 * slope(n)

gain_rise
Increase the slope when no saturation has been detected on the previous block of samples. The top limit of the slope is 1. Format is Q15 and 1 is added to get the coefficient of the multiplication.
Recommended value is 655
Ex: gain_rise = 655 => slope(n+1) = max(1, 1.02 * slope(n))

Information about Limiter parameters setting can be found in [6].

26.6.2.1.1.4.2 T_AUDIO_IIR_CFG

IIR filter replaces the FIR filter by having best performances using fewer coefficients. The aim of the IIR filter is to compensate the speaker frequency response in order to fit in ETSI requirements. Please refer to [3] for an overview of the module,

IIR filter is only available in TCS 3.x software except TCS 3.0. Other software versions do not include this structure in the speaker settings. When IIR is supported, the FIR filter isn't used so the FIR coefficients aren't included in the speaker settings.
IIR module only works in DAI acoustic and GSM path mode.

IIR settings are inside following structure of the speaker settings:

```
typedef struct
{
  BOOLEAN  iir_enable;
  UINT8    nb_iir_blocks;
  INT16    iir_coefs[80];
  UINT8    nb_fir_coefs;
  INT16    fir_coefs[32];
  INT8     input_scaling;
  INT8     fir_scaling;
  INT8     input_gain_scaling;
  INT8     output_gain_scaling;
  UINT16   output_gain;
  INT16    feedback;
```

```
}
T_AUDIO_IIR_CFG;
```

iir_enable
Enable/disable the IIR filter
0- disable
1- enable

In case of Read Access, the following parameters are valid only if iir_enable = 1.

nb_iir_blocks
Number of blocks for the given implementation of the systolic lattice IIR filter.
Value can be:
0 Recursive filtering part is disabled
1 Forbidden
[4:6] Number of IIR blocks

iir_coefs
Array containing the coefficients of the IIR lattice filter. There are 8 coefficients per block. The coefficients are generated by the MATLAB script sections.m. See [4] for more information.

nb_fir_coefs
Number of coefficients for the FIR (degree+1 of the FIR polynomial). Thus number must be greater or equal to 2 or the FIR filtering will be removed. It must be **lower or equal to 32**.

fir_coefs
Array containing the coefficients of the FIR filter. First coefficient (index 0) is the last coefficient corresponding to the term of higher degree in the FIR polynomial.

input_scaling
Used to scale the input at entry of the IIR to avoid overflows inside the IIR.
Note that the output of the filter (after the FIR) is scaled in the opposite way to compensate for this initial scaling.
The scaling is in $[-16,15]$ range.

fir_scaling
Used to scale the output of the IIR before using the FIR to avoid any scaling in the FIR. The output of the filter is scaled in the opposite way to compensate for this temporary scaling.
The scaling is in $[-16,15]$ range.

input_gain_scaling
The scaling factor applied at input of the filter for the global gain. Useful if the gain to implement is lower than 1 and if there are some overflows in the filter.
The scaling is in $[-16,15]$ range.

output_gain_scaling
Scaling factor at the output of the filter for the gain. Useful if the global gain to implement is higher than 1.
The scaling is in $[-16,15]$ range.

output_gain
A gain between [0,2[to tune the value of the global gain applied by the module. Format is unsigned Q15.

feedback
Used to tune the rounding noise of the IIR implementation and to remove the bias. This value is filter dependent and should be tuned for a given set of IIR coefficients. Format is Q15.

Information about IIR parameters setting can be found in [4].

26.6.2.1.1.5 T_AUDIO_STEREO_SPEAKER_SETTING

Specifies the characteristic of the speaker audio stereo path.

```
typedef struct
{
  /* mode of the speaker */
```

```
    INT8                        mode;
    /* Setting of the current mode */
    T_AUDIO_STEREO_SPEAKER_MODE setting;
  }
 T_AUDIO_STEREO_SPEAKER_SETTING;
```

```
typedef union
    {
     /* headphone mode parameters */
     T_AUDIO_STEREO_SPEAKER_MODE_HEADPHONE headphone;
     /* handheld mode parameters */
     T_AUDIO_STEREO_SPEAKER_MODE_HANDHELD  handheld;
     /* handfree mode parameters */
     T_AUDIO_STEREO_SPEAKER_MODE_HANDFREE  handfree;
     T_AUDIO_STEREO_SPEAKER_MODE_AUX              aux;
     T_AUDIO_STEREO_SPEAKER_MODE_CARKIT   carkit;
    }
    T_AUDIO_STEREO_SPEAKER_MODE;

typedef struct
    {
     /* stereo/mono configuration of the speaker */
     INT8   stereo_mono;
     /* sampling rate frequency */
     INT8   sampling_frequency;
    }
    T_AUDIO_STEREO_SPEAKER_MODE_HEADPHONE;

    typedef struct
    {
     /* sampling rate frequency */
     INT8   sampling_frequency;
    }
    T_AUDIO_STEREO_SPEAKER_MODE_HANDHELD;

    typedef struct
    {
     /* sampling rate frequency */
     INT8   sampling_frequency;
    }
    T_AUDIO_STEREO_SPEAKER_MODE_HANDFREE;

    typedef struct
    {
     /* sampling rate frequency */
     INT8   sampling_frequency;
    }
    T_AUDIO_STEREO_SPEAKER_MODE_AUX;

    typedef struct
    {
     /* stereo/mono configuration of the speaker */
     INT8   stereo_mono;
     /* sampling rate frequency */
     INT8   sampling_frequency;
    }

    T_AUDIO_STEREO_SPEAKER_MODE_CARKIT;
```

Mode

Specifies the mode of the microphone: **AUDIO_STEREO_SPEAKER_MODE_HEADPHONE**, **AUDIO_STEREO_SPEAKER_MODE_HANDHELD**, **AUDIO_STEREO_SPEAKER_MODE_HANDFREE**, **AUDIO_STEREO_SPEAKER_MODE_AUX**, **AUDIO_STEREO_SPEAKER_MODE_CARKIT**	
AUDIO_STEREO_SPEAKER_MODE_HEADPHONE ***mode: this mode is only available with the analog base band SYREN.***	
	stereo_mono: Specifies the possible stereo-mono conversion: - AUDIO_STEREO (no conversion) - AUDIO_MONO_LEFT (convert to mono and transmit on left channel) - AUDIO_MONO_RIGHT (convert to mono and transmit on right channel) - AUDIO_MONO_LEFT | AUDIO_MONO_RIGHT (convert to mono and transmit on both channels)
	sampling_frequency: Specifies the audio stereo sampling rate frequency - AUDIO_STEREO_SAMPLING_FREQUENCY_48KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_44_1KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_32KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_24KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_22_05KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_16KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_12KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_11_025KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_8KHZ **WARNING:** sampling frequency can not be changed after PLL power on.
AUDIO_STEREO_SPEAKER_MODE_HANDHELD ***mode:***	
	sampling_frequency: See above
	Note: stereo-mono conversion is set to AUDIO_MONO_LEFT.
AUDIO_STEREO_SPEAKER_MODE_HANDFREE ***mode:***	
	sampling_frequency: See above
	Note: stereo-mono conversion is set to AUDIO_MONO_LEFT.
AUDIO_STEREO_SPEAKER_MODE_AUX ***mode:***	
	sampling_frequency: See above
	Note: stereo-mono conversion is set to AUDIO_MONO_LEFT.
AUDIO_STEREO_SPEAKER_MODE_CARKIT ***mode:***	
	stereo_mono: See above
	sampling_frequency: See above

26.6.2.1.2 T_AUDIO_MICROPHONE_SPEAKER_LOOP_SETTING

Specifies the characteristic of the features involved in the loop between the speaker and the microphone. **For DSP codes >= 33, there is a new version of AEC called NEW AEC**:

```
typedef struct
{
  /* gain of the sidetone */
  INT16                         sidetone_gain;
  /* configuration of the acoustic echo cancellation */
  T_AUDIO_AEC_CFG               aec;
}
 T_AUDIO_MICROPHONE_SPEAKER_LOOP_SETTING;
```

Sidetone

Specifies the gain in 3 dB unit to add to the loop between the microphone and the speaker. The range is from –23 dB to 1 dB (3 dB by 3 dB). Note if the variable is equal to AUDIO_SIDETONE_OPEN, there's no loop between the microphone and the speaker.

This mode is only available in GSM and bluetooth cordless voice path mode.

WARNING: IF THE SPEAKER FILTER IS BYPASSED, THE SIDETONE IS OPEN.

AEC

aec_enable

Specifies if the AEC module must be enabled (AUDIO_AEC_ENABLE) or disabled (AUDIO_AEC_DISABLE).

Note: AEC is only available in GSM and all DAI mode.

In case of Read Access, the following parameters are valid only if anr_enable = 1.

aec_mode

Specifies the mode of the cancellation: AUDIO_SHORT_ECHO: short echo cancellation, AUDIO_LONG_ECHO: long echo cancellation.

Note: AEC is only available in GSM all DAI mode.

echo_suppression_level

Specifies the additional echo suppression level.

Note: noise suppression is only available in GSM and all mode.

Level name	level (dB)
AUDIO_ECHO_0dB	0
AUDIO_ECHO_6dB	6
AUDIO_ECHO_12dB	12
AUDIO_ECHO_18dB	18

noise_enable

Specifies if the noise suppression module must be enable (AUDIO_NOISE_SUPPRESSION_ENABLE) or disable (AUDIO_NOISE_SUPPRESSION_DISABLE).

Note: noise suppression is only available in GSM and all DAI mode.

Note: The noise suppressor is replaced by ANR in TCS3.x except 3.0.

noise_suppression_level

Specifies the noise suppression limitation level.

Note: AEC is only available in GSM and all DAI mode.

Level name	level (dB)
AUDIO_NOISE_NO_LIMIT	no limitation
AUDIO_NOISE_6dB	-6
AUDIO_NOISE_12dB	-12
AUDIO_NOISE_18dB	-18

Note: The noise suppressor is replaced by ANR in TCS3.x except 3.0.

NEW AEC (detailed format of the parameters can be found in [2])

aec_enable

Specifies if the AEC module must be enabled (AUDIO_AEC_ENABLE) or disabled (AUDIO_AEC_DISABLE).

Note: NEW AEC is only available in GSM and all DAI mode.

In case of Read Access, the following parameters are valid only if aec_enable = 1.

continuous_filtering

Enable (TRUE) or disable (FALSE) continuous mode filtering.

granularity_attenuation

granularity of the smoothed attenuation.

smoothing_coefficient

smoothing coefficient.

max_echo_suppression_level

maximum attenuation level. Some values are defined as constants:
AUDIO_MAX_ECHO_xdB with x being 0, 2, 3, 6, 12, 18, 24.

vad_factor

VAD factor relative to the current estimated energy.

absolute_threshold

VAD absolute offset relative to the current estimated energy.

factor_asd_filtering

modifying factor of d_far_end_noise for filtering decision.

factor_asd_muting

modifying factor of d_far_end_noise for muting decision.

aec_visibility
Enable (AUDIO_AEC_VISIBILITY_ ENABLE) or disable (AUDIO_AEC_VISIBILITY_ DISABLE) AEC visibility. A copy of far_end_pow and far_end_noise is traced in Layer1 every SC_AEC_VISIBILITY_INTERVAL frames. It is intended for debug purposes and can only be disabled by a new AEC request (i.e. going back to idle mode won't disable visibility for next call).

noise_enable
Specifies if the noise suppression module must be enabled (AUDIO_NOISE_SUPPRESSION_ENABLE) or disabled (AUDIO_NOISE_SUPPRESSION_DISABLE).
Note: The noise suppressor is replaced by ANR in TCS3.x except 3.0.

noise_suppression_level
Specifies the noise suppression limitation level.
Note: AEC is only available in GSM and all DAI mode.

Level name	level (dB)
AUDIO_NOISE_NO_LIMIT	no limitation
AUDIO_NOISE_6dB	-6
AUDIO_NOISE_12dB	-12
AUDIO_NOISE_18dB	-18

Note: The noise suppressor is replaced by ANR in TCS3.x except 3.0.

26.6.2.1.2.1 T_AUDIO_MICROPHONE_SPEAKER_SETTING

Specifies the characteristic of the features that are common to the speaker and the microphone.

```
typedef struct
 {
  /* volume speed control */
  INT16 volume_speed;
  /* audio on/off */
  INT8  audio_onoff;
 }
 T_AUDIO_MICROPHONE_SPEAKER_SETTING;
```

Below the detail of each parameter:

volume_speed ***(only available in case of non-TI audio ABB used with P2 samples)***
speed to change the volume in downlink and in uplink
values are from 0x0001 (low speed) to 0x7FFF (high speed) in signed Q15 format, ex:
0x1 → (2^15)/1 = 32768 samples to reach the volume level
0x2 → (2^15)/2 = 16384 samples to reach the volume level
0x7FFF → (2^15)/32767 = 1 sample to reach the volume level

Audio_onoff ***(only available on Calypso+ and Perseus 2 samples)***
If set to 1, it starts ABB audio and disable the automatic stop when no DSP audio activity is running.
If set to 0, it will stop ABB audio when there is no DSP audio activity running.

26.7 Full Access Family

This section describes all the API functions belong to the full access family.

26.7.1 API Functions

26.7.1.1 audio_full_access_write

```
T_AUDIO_RET audio_full_access_write (
                        T_AUDIO_FULL_ACCESS_WRITE     *p_parameter,
                        T_RV_RETURN                    return_path)
```

Description

This function is called to configure any value belonging to the audio mode structure.

Parameters

- **T_AUDIO_FULL_ACCESS_WRITE**

```
typedef struct {
  UINT8   variable_indentifier;
  // identifier of the variable to configure
  void    *data;
  // data corresponding to the variable to set
}T_AUDIO_FULL_ACCESS_WRITE;
```

Below the detail of each parameters (note for the description of the data please see the MMI family chapter):

Identifier	Associated data format
AUDIO_PATH_USED	typedef UINT8 T_AUDIO_VOICE_PATH_SETTING;
AUDIO_MICROPHONE_MODE *(1)*	INT8
AUDIO_MICROPHONE_GAIN *(2)*	INT8
AUDIO_MICROPHONE_EXTRA_GAIN *(1)*	INT8
AUDIO_MICROPHONE_OUTPUT_BIAS *(1)*	INT8

AUDIO_MICROPHONE_FIR	typedef struct
	{
	UINT16 coefficient[31];
	}
	T_AUDIO_FIR_COEF;

AUDIO_MICROPHONE_ANR *(6)*	T_AUDIO_ANR_CFG (see 28.6.2.1.1.3.1)
AUDIO_MICROPHONE_ES *(6)*	T_AUDIO_ES_CFG (see 28.6.2.1.1.3.2)
AUDIO_SPEAKER_MODE *(1)*	INT8
AUDIO_SPEAKER_GAIN *(2)*	INT8
AUDIO_SPEAKER_EXTRA_GAIN *(1)*	INT8
AUDIO_SPEAKER_FILTER *(1)*	INT8
AUDIO_SPEAKER_HIGHPASS_FILTER *(1)*	INT8

AUDIO_SPEAKER_FIR *(5)*	typedef struct
	{
	UINT16 coefficient[31];
	}
	T_AUDIO_FIR_COEF;

AUDIO_SPEAKER_IIR *(6)*	T_AUDIO_IIR_CFG (see **Error! Reference source not found.**)
AUDIO_SPEAKER_LIMITER *(6)*	T_AUDIO_LIMITER_CFG (see **Error! Reference source not found.**)
AUDIO_SPEAKER_BUZZER *(1)*	INT8
AUDIO_MICROPHONE_SPEAKER_LOOP_SIDETONE *(2)*	INT8

AUDIO_MICROPHONE_SPEAKER_LOOP_AEC	typedef struct
	{
	/* Enable the AEC */
	UINT16 aec_enable;
	/* Mode of the AEC */
	UINT16 aec_mode;
	/* level of the echo cancellation */
	UINT16 echo_suppression_level;
	/* enable the noise suppression */
	UINT16 noise_suppression_enable;
	/* level of the noise suppression */
	UINT16 noise_suppression_level;
	}
	T_AUDIO_AEC_CFG;

	noise_suppression_level and noise_suppression_enable not available in TCS3.0 software except TCS3.0 (replaced by ANR)
AUDIO_STEREO_SPEAKER_MODE *(1)*	INT8
AUDIO_STEREO_SPEAKER_STEREO_MONO *(1)*	INT8
AUDIO_STEREO_SPEAKER_SAMPLING_FREQUENCY *(1)*	INT8

AUDIO_SPEAKER_VOLUME_LEVEL *(1)*	typedef struct
	{
	/* volume of the audio speaker */
	UINT8 audio_speaker_level;
	}
	T_AUDIO_SPEAKER_LEVEL;

AUDIO_STEREO_SPEAKER_VOLUME_LEVEL *(1)*	typedef struct
	{
	/* volume of the audio speaker */
	UINT8 audio_stereo_speaker_level_left;
	UINT8 audio_stereo_speaker_level_right;
	}
	T_AUDIO_STEREO_SPEAKER_LEVEL;

AUDIO_ONOFF *(3)*	INT8
AUDIO_VOLUME_SPEED *(4)*	INT16

(1) Not available when using a non-TI ABB for audio tasks
(2) When using a non-TI ABB for audio tasks, type of parameter is Int16 instead of Int8
(3) Only available on Calypso+ and Perseus2 samples
(4) Only available when using a non-TI ABB for audio tasks
(5) Not available in TCS3.x software except TCS3.0
(6) Only available in TCS3.x software except TCS3.0

- **T_RV_RETURN**

C.f. API function *audio_mode_load*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_mode_load*.

Event Return

- **AUDIO_FULL_ACCESS_WRITE_DONE**

This event informs that the value was written.

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
}T_AUDIO_FULL_ACCESS_WRITE_DONE;
```

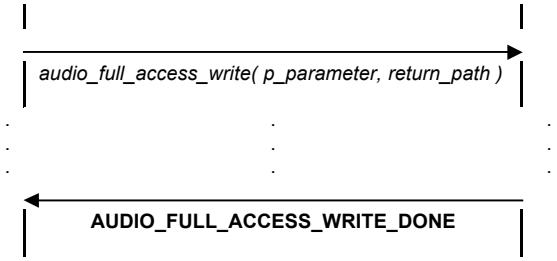
The possible values of *status* are:

value	Id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped
-1	AUDIO_ERROR	A problem occurs during the writing process..

Current restriction of use

None.

Process flow



26.7.1.2 audio_full_access_read

```
T_AUDIO_RET audio_full_access_read (
                        T_AUDIO_FULL_ACCESS_READ *p_parameter)
```

Description

This function is called to read any value belonging to the audio mode structure.

Parameters

- **T_AUDIO_FULL_ACCESS_READ**

```
typedef struct {
  UINT8  variable_indentifier; // identifier of the variable to read
  VOID   *data                 // data to return
}T_AUDIO_FULL_ACCESS_READ;
```

Below the detail of each parameters (note for the description of the data please see the MMI family chapter):

Identifier	Associated data format
AUDIO_PATH_USED	typedef UINT8 T_AUDIO_VOICE_PATH_SETTING;
AUDIO_MICROPHONE_MODE *(1)*	INT8
AUDIO_MICROPHONE_GAIN *(2)*	INT8
AUDIO_MICROPHONE_EXTRA_GAIN *(1)*	INT8
AUDIO_MICROPHONE_OUTPUT_BIAS *(1)*	INT8

AUDIO_MICROPHONE_FIR	typedef struct
	{
	UINT16 coefficient[31];
	}
	T_AUDIO_FIR_COEF;

AUDIO_MICROPHONE_ANR *(6)*	T_AUDIO_ANR_CFG (see *28.6.2.1.1.3.1*)
AUDIO_MICROPHONE_ES *(6)*	T_AUDIO_ES_CFG (see *28.6.2.1.1.3.2*)
AUDIO_SPEAKER_MODE *(1)*	INT8
AUDIO_SPEAKER_GAIN *(2)*	INT8
AUDIO_SPEAKER_EXTRA_GAIN *(1)*	INT8
AUDIO_SPEAKER_FILTER *(1)*	INT8
AUDIO_SPEAKER_HIGHPASS_FILTER *(1)*	INT8

AUDIO_SPEAKER_FIR *(5)*	typedef struct
	{
	UINT16 coefficient[31];
	}
	T_AUDIO_FIR_COEF;

AUDIO_SPEAKER_IIR *(6)*	T_AUDIO_IIR_CFG (see 28.6.2.1.1.4.2)
AUDIO_SPEAKER_LIMITER *(6)*	T_AUDIO_LIMITER_CFG (see 28.6.2.1.1.4.1)
AUDIO_SPEAKER_BUZZER *(1)*	INT8
AUDIO_MICROPHONE_SPEAKER_LOOP_SIDETONE	INT8

(2)	

AUDIO_MICROPHONE_SPEAKER_LOOP_AEC	typedef struct
	{
	/* Enable the AEC */
	UINT16 aec_enable;
	/* Mode of the AEC */
	UINT16 aec_mode;
	/* level of the echo cancellation */
	UINT16 echo_suppression_level;
	/* enable the noise suppression */
	UINT16 noise_suppression_enable;
	/* level of the noise suppression */
	UINT16 noise_suppression_level;
	}
	T_AUDIO_AEC_CFG;
	
	noise_suppression_level and noise_suppression_enable not available in TCS3.0 software except TCS3.0 (replaced by ANR)

AUDIO_STEREO_SPEAKER_MODE *(1)*	UINT8
AUDIO_STEREO_SPEAKER_STEREO_MONO *(1)*	UINT8
AUDIO_STEREO_SPEAKER_SAMPLING_FREQUENCY *(1)*	INT8

AUDIO_SPEAKER_VOLUME_LEVEL *(1)*	typedef struct
	{
	/* volume of the audio speaker */
	UINT8 audio_speaker_level;
	}
	T_AUDIO_SPEAKER_LEVEL;

AUDIO_STEREO_SPEAKER_VOLUME_LEVEL *(1)*	typedef struct
	{
	/* volume of the audio speaker */
	UINT8 audio_stereo_speaker_level;
	}
	T_AUDIO_STEREO_SPEAKER_LEVEL;

AUDIO_ONOFF *(3)*	INT8
AUDIO_VOLUME_SPEED *(4)*	INT16

(1) Not available when using a non-TI ABB for audio tasks
(2) When using a non-TI ABB for audio tasks, type of parameter is Int16 instead of Int8
(3) Only available on Calypso+ and Perseus2 samples
(4) Only available when using a non-TI ABB for audio tasks
(5) Not available in TCS3.x software except TCS3.0
(6) Only available in TCS3.x software except TCS3.0

- **T_RV_RETURN**

C.f. API function *audio_mode_load*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_mode_load*.

- Moreover the data pointer of the T_AUDIO_FULL_ACCESS_READ structure points to the data to returned data if the *T_AUDIO_RET* value is equal to *AUDIO_OK*.

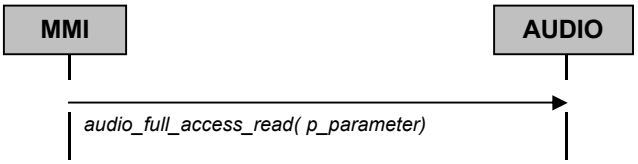
Event Return

None.

Current restriction of use

None.

Process flow



Chapter 27 Memory Card

27.1 Introduction

This section describes the Memory Card(TFlash Driver) APIs used for Locosto.

27.2 Interface description

27.2.1 mc_subscribe

```
T_RV_RET mc_subscribe (T_MC_SUBSCRIBER *subscriber_p
                       T_RV_RETURN return_path)
```

Description

This functions can be used by a client to subscribe to the MC-driver. The client shall provide a return path which will be used for all functions returning an event. The client will receive an MC_SUBSCRIBE_RSP_MSG response event from the driver indicating the result of the subscription request. If the result is successful the client is able to use the driver services like reading and writing data.

Parameters

- **subscriber_p**

Subscriber identification value, which shall be allocated by the client and is filled by the driver.

- **return_path**

The return path of the client.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request

Event Return

MC_SUBSCRIBE_RSP_MSG event is returned to the calling SWE.

Current restriction of use

None.

27.2.2 mc_unsubscribe

```
T_RV_RET mc_unsubscribe (T_MC_SUBSCRIBER *subscriber_p)
```

Description

This function can be used by a client to unsubscribe from the driver. The client will receive an MC_UNSUBSCRIBE_RSP_MSG response event from the driver indicating the result of the un-

subscription request. If the result is successful the T_MC_SUBSCRIBER area is no longer useful and can be safely reused by the client.
The driver will handle each request sequentially. Pending requests after a un-subscription, means that the client has issued the requests after the un-subscription request. These messages will be discarded. The client has to re-subscribe if it wants to send requests again.

Parameters

- **subscriber_p**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request.
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

MC_UNSUBSCRIBE_RSP_MSG event is returned to the calling SWE.

Current restriction of use

None.

27.2.3 mc_read

```
T_RV_RET mc_read (T_MC_RCA rca, T_MC_RW_MODE mode,
                  UINT32 addr, UINT8 *data_p,
                  UINT32 data_size,
                  T_MC_SUBSCRIBER subscriber)
```

Description

This function reads data from an MMC/SD-card using a specific transfer mode. If partial reads are allowed (if CSD parameter READ_BL_PARTIAL is set) the start address can start and stop at any address within the card address space, otherwise it shall start and stop at block boundaries. The client is responsible for setting the correct address and data size parameter according to the device properties. The client can obtain these properties by reading the CSD-register.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected MMC/SD-cards can be read using the *mc_read_card_stack()* function.

- **mode**

Indicate the desired read method.

- MC_RW_STREAM
- MC_RW_BLOCK

- **addr**

The physical start address in bytes units from where to read data.

- **data_p**

Pointer to a destination buffer, provided by the client, where the driver will put the data. The buffer size shall be at least *data_size* bytes.

- **data_size**

Number of bytes to be read from the card.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

MC_ READ_RSP_MSG event is returned to the calling SWE.

Current restriction of use

Only block oriented data transfer (MC_RW_BLOCK) is supported at this moment.

27.2.4 mc_write

```
T_RV_RET mc_write (T_MC_RCA rca, T_MC_RW_MODE mode,
                   UINT32 addr, UINT8 *data_p,
                   UINT32 data_size,
                   T_MC_SUBSCRIBER subscriber)
```

Description

This function writes data to an MMC/SD-card using a specific transfer mode. If partial reads are allowed (if CSD parameter WRITE_BL_PARTIAL is set) the start address can start and stop at any address within the card address space, otherwise it shall start and stop at block boundaries. The client is responsible for setting the correct address and data size parameter according to the device properties. The client can obtain these properties by reading the CSD-register.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected MMC/SD-cards can be read using the *mc_read_card_stack()* function.

- **mode**

Indicate the desired write method.

- MC_RW_STREAM
- MC_RW_BLOCK

- **addr**

The physical start address in bytes units from where to read data.

- **data_p**

Pointer to a source buffer, provided by the client, from where the driver will read the data. The buffer size shall be at least *data_size* bytes.

- **data_size**

Number of bytes to be write to the card.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

MC_ WRITE_RSP_MSG event is returned to the calling SWE.

Current restriction of use

- This service is not available for MMC/SD ROM cards.
- Only SD cards provide a mechanical write protect switch. If the write protect switch of the SD-card is set to write protect, this service is not available.

In the above cases, the MC driver will respond with a RV_NOT_SUPPORTED in the write response message.

- Only block oriented data transfer (MC_RW_BLOCK) is supported at this moment.

27.2.5 mc_erase

```
T_RV_RET mc_erase (T_MC_RCA rca, UINT32 erase_group_start,
                   UINT32 erase_group_end, T_MC_SUBSCRIBER subscriber)
```

Description

This function erases a range of erase groups on the card. The size of the erase group is specified in the CSD. The erase group start and end address is given in bytes units. This address will be rounded down to the erase group boundary.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected MMC/SD-cards can be read using the *mc_read_card_stack()* function.

- **start_group**

Erase group address in bytes units where erasing will start.

- **end_group**

Erase group address in bytes units where erasing will end.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.

RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

MC_ERASE_GROUP_MSG event is returned to the calling SWE.

Current restriction of use

- This service is not available for MMC/SD ROM cards.
- If the write protect switch of the SD-card is set to write protect, this service is not available.

In the above cases, the MC driver will respond with a RV_NOT_SUPPORTED in the write response message.

27.2.6 mc_set_write_protect

```
T_RV_RET mc_set_write_protect (T_MC_RCA rca, UINT32 wr_prot_group,
                               T_MC_SUBSCRIBER subscriber)
```

Description

This function sets the write protection of the addressed write protect group against erase or write. The group size is defined in units of WP_GRP_SIZE erase group as specified in the CSD.
This function does not write protect the entire card which can be done by setting the permanent or temporary write protect bits in the CSD. For this the ***Error! Reference source not found.****()* function shall be used.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected MMC/SD-cards can be read using the *mc_read_card_stack()* function.

- **wr_prot_group**

The group address in byte units. The LSB's below the group size will be ignored.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

MC_SET_WRITE_PROTECT_RSP_MSG event is returned to the calling SWE.

Current restriction of use

This service is not available for MMC/SD ROM cards.

27.2.7 mc_clr_write_protect

```
T_RV_RET mc_clr_write_protect (T_MC_RCA rca, UINT32 wr_prot_group,
```

```
                                    T_MC_SUBSCRIBER subscriber)
```

Description

This function clears the write protection of the addressed write protect group. The group size is defined in units of WP_GRP_SIZE erase group as specified in the CSD.
This function does not disable write protect of the entire card which can be done by erasing the temporary write protect bits in the CSD. For this the ***Error! Reference source not found.()*** function shall be used.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected MMC/SD-cards can be read using the *mc_read_card_stack()* function.

- **wr_prot_group**

The group address in byte units. The LSB's below the group size will be ignored.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

MC_CLR_PROTECT_RSP_MSG event is returned to the calling SWE.

Current restriction of use

This service is not available for MMC/SD ROM cards.

27.2.8 mc_get_write_protect

```
T_RV_RET mc_get_write_protect (T_MC_RCA rca, UINT32 wr_prot_group,
                               T_MC_SUBSCRIBER subscriber)
```

Description

This function reads 32 write protection bits representing 32 write protect groups starting at the specified address.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected MMC/SD-cards can be read using the *mc_read_card_stack()* function.

- **wr_prot_group**

The group address in byte units. The LSB's below the group size will be ignored.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

MC_GET_PROTECT_RSP_MSG event is returned to the calling SWE.

Current restriction of use

This service is not available for MMC/SD ROM cards.

27.2.9 mc_get_card_status

```
T_RV_RET mc_get_card_status(T_MC_RCA rca, T_MC_SUBSCRIBER subscriber)
```

Description

This function returns the 32-bit status register of the MMC/SD-card. This status is not buffered in the driver but will be read directly from the card. See [MMC], paragraph 4.10 for an explanation of the status bits.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected MMC/SD-cards can be read using the *mc_read_card_stack()* function.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

T_MC_CARD_STATUS_RSP_MSG is returned.

Current restriction of use

None

27.2.10 mc_dma_mode

```
T_RV_RET mc_dma_mode(T_MC_DMA_MODE dma_mode)
```

Description

This function selects the DMA mode to be used by the driver. Default setting will be MC_DMA_AUTO.

Parameters

- **dma_mode**

Indicates whether to use DMA or let the CPU handle the copying. Possible values:

- MC_FORCE_CPU (Use CPU to transfer data to RAM)
- MC_FORCE_DMA (Use DMA to transfer data to RAM)
- MC_DMA_AUTO (Driver determines CPU or DMA transfer)

Immediate Return

- **T_RV_RET**

The possible values are:

id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_INVALID_PARAMETER	invalid mode.

Event Return

None

Current restriction of use

None

27.2.11 mc_update_acq

```
T_RV_RET mc_update_acq (T_MC_SUBSCRIBER subscriber)
```

Description

This function starts an identification cycle of a card stack (acquisition procedure). The card management information in the controller will be updated. New cards will be initialised; old cards keep their configuration. At the end all active cards are in Stand-by state.

After this function has completed the number of cards connected can be retrieved with the *mc_get_stack_size()* function. The session address of each connected card can be retrieved with the *mc_read_card_stack()* function.

Parameters

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_INVALID_PARAMETER	subscriber is invalid
RV_MEMORY_ERR	Insufficient memory to create message request

Event Return

MC_UPDATE_ACQ_RSP MSG event is returned to the calling SWE.

Current restriction of use

Only 1 MMC/SD card supported at this moment.

27.2.12 mc_reset

```
T_RV_RET mc_reset(T_MC_SUBSCRIBER subscriber)
```

Description

This function resets all cards to idle state. This function executes the GO_IDLE_STATE command (CMD0) on the bus. After completion of this service the mc_update_acq() function shall be called before the MMC/SD-cards can be used.

Parameters

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_INVALID_PARAMETER	subscriber is invalid
RV_MEMORY_ERR	Insufficient memory to create message request

Event Return

MC_RESET_RSP_MSG event is returned to the calling SWE.

Current restriction of use

None.

27.2.13 mc_get_stack_size

```
T_RV_RET mc_get_stack_size (UINT16 *size_p)
```

Description

This function returns the number of connected MMC/SD-cards.

Parameters

- **size_p**

Pointer to integer value allocated by the client, in which the driver stores the stack size.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_INVALID_PARAMETER	size_p is NULL.

Event Return

Not applicable

Current restriction of use

None.

27.2.14 mc_read_card_stack

```
T_RV_RET mc_read_card_stack (T_MC_RCA *stack_p, UINT16 size)
```

Description

This function returns the relative card address of each individual MMC/SD-card on the MMC/SD-bus. The client needs to provide an array of T_MC_RCA. The size of the array can be determined with the *mc_get_stack_size()* function.

Parameters

- **stack_p**

Pointer to T_MC_RCA array.

- **size**

Array size in units of T_MC_RCA.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_INVALID_PARAMETER	-stack_p is NULL. -size is too small.

Event Return

None.

Current restriction of use

At this moment only 1 MMC/SD card is supported.

27.2.15 mc_read_OCR

```
T_RV_RET mc_read_OCR (T_MC_RCA rca, UINT32 *ocr_p,
                      T_MC_SUBSCRIBER subscriber)
```

Description

This function returns the 32-bit OCR-register from an MMC/SD-card. This register is not buffered in the driver and therefore will be read directly from the card.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected MMC/SD-cards can be read using the *mc_read_card_stack()* function.

- **ocr_p**

Pointer to an 32-bits data location, provided by the client, to which the driver copies the OCR.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

MC_READ_OCR_RSP_MSG event is returned to the calling SWE.

Current restriction of use

None.

27.2.16 mc_read_CID

```
T_RV_RET mc_read_CID (T_MC_RCA rca, UINT8 *cid_p,
                      T_MC_SUBSCRIBER subscriber)
```

Description

This function returns the 128-bit CID register from a MMC/SD-card. This register is not buffered in the driver and therefore will be read directly from the card.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected MMC/SD-cards can be read using the *mc_read_card_stack()* function.

- **cid_p**

Pointer to a 128-bit buffer, allocated by the client, to which the driver copies the CID. cid_p points to the LSB of the CID.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

MC_READ_CID_RSP_MSG event is returned to the calling SWE.

Current restriction of use

None.

27.2.17 mc_read_CSD

```
T_RV_RET mc_read_CSD (T_MC_RCA rca, UINT8 *csd_p,
                      T_MC_SUBSCRIBER subscriber)
```

Description

This function returns the 128-bit CSD from a MMC/SD-card. This register is not buffered in the driver and therefore will be read directly from the card.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected MMC/SD-cards can be read using the *mc_read_card_stack()* function.

- **csd_p**

Pointer to a 128-bit buffer, allocated by the client, to which the driver copies the CSD. csd_p points to the LSB of the CSD.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

MC_READ_CSD_RSP_MSG event is returned to the calling SWE.

Current restriction of use

None.

27.2.18 mc_sd_read_SCR

```
T_RV_RET mc_sd_read_scr (T_MC_RCA rca, UINT8 *scr_p,
                         T_MC_SUBSCRIBER subscriber)
```

Description

This function returns the 64-bit SCR (SD Card Configuration Register) from a SD card. This register is not buffered in the driver and therefore will be read directly from the card.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected SD card can be read using the *mc_read_card_stack()* function.

- **scr_p**

Pointer to a buffer of 64-bits, allocated by the client, to which the driver copies the SCR. scr_p points to the LSB of the SCR.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

MC_READ_SCR_RSP_MSG event is returned to the calling SWE.

Current restriction of use

None.

27.2.19 mc_get_sw_version

```
UINT32 mc_get_sw_version (void)
```

Description

This function returns the software version of the driver. The version is a 32-bit value which is organised as follows:

major version number (8 bits)		minor version number (8bits)		build number (16 bits)	
31	24	23	16	15	0

Parameters

Not applicable.

Immediate Return

- **UINT32**

The 32-bit software version

Event Return

Not applicable.

Current restriction of use

None.

27.2.20 mc_get_card_type

```
T_MC_CARD_TYPE mc_get_card_type (T_MC_RCA rca)
```

Description

This function returns the card type.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected MMC/SD-cards can be read using the *mc_read_card_stack()* function.

Immediate Return

- **T_MC_CARD_TYPE**

Card type at requested Relative Card Address.
The possible values are:

Id	Definition
NO_CARD	No card available at given RCA
SD_CARD	SD card present on given RCA
MMC_CARD	MMC card present on given RCA

Event Return

Not applicable.

Current restriction of use

None.

27.2.21 mc_sd_get_card_status

```
T_RV_RET mc_sd_get_card_status (T_MC_RCA rca, UINT8 *sd_status_p,
                                T_MC_SUBSCRIBER subscriber)
```

Description

This function returns the 512-bit SD status register of the SD-card. This status is not buffered in the driver but will be read directly from the card. See [SD], paragraph 4.10.2 for an explanation of the status bits.

Parameters

- **rca**

Relative Card Address. This relative card address of the connected SD-cards can be read using the *mc_read_card_stack()* function.

- sd_status_p

Pointer to a 512-bit buffer, allocated by the client, to which the driver copies the SD status. sd_status_p points to the LSB of the status.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

T_MC_SD_CARD_STATUS_RSP_MSG is returned.

Current restriction of use

None

27.2.22 mc_send_notification

```
T_RV_RET mc_send_notification (T_MC_EVENT event, T_MC_SUBSCRIBER subscriber)
```

Description

This function is used to inform the MC driver the subscriber wants to be notified if an MMC/SD event occurs. The currently supported events are for card insertion and card removal detection. After an event occurs, the MC driver will send a notification to all subscribers that requested it.

Parameters

- **event**

A combination of events (logical OR) to which the subscriber wants to be notified.

- **subscriber**

Subscriber identification value.

Immediate Return

- **T_RV_RET**

The possible values are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is not initialised.
RV_MEMORY_ERR	Insufficient memory to create message request
RV_INVALID_PARAMETER	subscriber is invalid

Event Return

T_MC_NOTIFICATION_RSP_MSG is returned. If the response message indicates that the notification request has succeeded, the client can expect an event notification indication from the MC driver.

T_MC_EVENT_IND_MSG is send to the client when an event occurs.

Current restriction of use

None

27.3 Message definition

In this paragraph all messages are described. The response messages will return the result of the command and most of the time the content of the card status register (i.e. if the used MMC/SD-commands have a R1 response type). The RV_INTERNAL_ERROR result will be set if at least one error in the card status register is set.

The driver will set the RV_INVALID_PARAMETER response if one of parameter values is invalid. Example situations:

- Invalid RCA. The current card-stack does not contain an MMC/SD-card with this address;
- provided pointer value is NULL;
- etc.

Note: the parameter 'subscriber' is checked in the API functions not to be a NULL pointer. So invalid 'subscriber' parameter results in an API-function return value of RV_INVALLID_PARAMETER and does not result in a response message with status RV_INVALLID_PARAMETER.

27.3.1 Subscribe

The T_MC_SUBSCRIBE_REQ_MSG message can be used to subscribe to the MC-driver. This message is similar to the *mc_subscribe()* function (see 27.2.1). The driver responds with a T_MC_SUBSCRIBE_RSP_MSG message.

27.3.1.1 T_MC_SUBSCRIBE_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_SUBSCRIBER   *subscriber_p;
      T_RV_RETURN       return_path;
} T_MC_SUBSCRIBE_REQ_MSG
```

27.3.1.2 T_MC_SUBSCRIBE_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      T_MC_SUBSCRIBER   *subscriber_p;
} T_MC_SUBSCRIBE_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_INVALID_PARAMETER	subscriber_p is NULL
RV_INTERNAL_ERROR	-There was an internal error while executing the request. Execution was unsuccessful; -Maximum number of clients reached;

27.3.2 Unsubscribe

The T_MC_UNSUBSCRIBE_REQ_MSG message can be used to unsubscribe from the MC-driver. This message is similar to the *mc_unsubscribe()* function (see 27.2.2). The driver responds with a T_MC_UNSUBSCRIBE_RSP_MSG message.

27.3.2.1 T_MC_UNSUBSCRIBE_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_SUBSCRIBER   *subscriber_p;
} T_MC_UNSUBSCRIBE_REQ_MSG
```

27.3.2.2 T_MC_UNSUBSCRIBE_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
} T_MC_UNSUBSCRIBE_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_INVALID_PARAMETER	-subscriber_p is NULL; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

27.3.3 Data read

The T_MC_READ_REQ_MSG message can be used to read data from an MMC/SD card. This message is similar to the mc_read() function (see 27.2.3). The driver responds with a T_MC_READ_RSP_MSG message.

27.3.3.1 T_MC_ READ_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_RCA          rca;
      UINT32            addr;
      UINT8             *data_p;
      UINT32            data_size;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_READ_REQ_MSG
```

27.3.3.2 T_MC_ READ_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      UINT32            card_status;
      UINT8             *data_p;
      UINT32            data_size;
} T_MC_READ_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -addr is invalid; -data_p is NULL; -data_size is invalid; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check the Card status

27.3.4 Data write

The T_MC_ WRITE_REQ_MSG message can be used to write data to an MMC/SD card. This message is similar to the mc_write() function (see 27.2.4). The driver responds with a T_MC_WRITE_RSP_MSG message.

27.3.4.1 T_MC_WRITE_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_RCA          rca;
      UINT32            addr;
      UINT8             *data_p;
      UINT32            data_size;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_WRITE_REQ_MSG
```

27.3.4.2 T_MC_WRITE_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      UINT32            card_status;
```

```
      UINT8             *data_p;
      UINT32            data_size;
} T_MC_WRITE_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -addr is invalid; -data_p is NULL; -data_size is invalid; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check the Card status
RV_NOT_SUPPORTED	The selected card is a ROM card or write protected.

27.3.5 Ease group

The T_MC_ERASE_GROUP_REQ_MSG message can be used to erase a range of erase groups on the card. This message is similar to the mc_erase_group () function (see 27.2.5). The driver responds with a T_MC_ERASE_GROUP_RSP_MSG message.

27.3.5.1 T_MC_ERASE_GROUP_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_RCA          rca;
      UINT32            erase_group_start;
      UINT32            erase_group_end;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_ERASE_GROUP_REQ_MSG
```

27.3.5.2 T_MC_ERASE_GROUP_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      UINT32            card_status;
} T_MC_ERASE_GROUP_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -erase_selection of erase groups; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check the Card status
RV_NOT_SUPPORTED	The selected card is a ROM card or write protected.

27.3.6 Set write protect

The T_MC_SET_PROTECT_REQ_MSG message can be used to set the write protection of the addressed write protect group. This message is similar to the mc_set_write_protect() function (see 27.2.6). The driver responds with a T_MC_SET_PROTECT_RSP_MSG message.

27.3.6.1 T_MC_SET_PROTECT _REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_RCA          rca;
      UINT32            wr_prot_group;
      T_MC_SUBSCRIBER   subscriber;
```

```
} T_MC_SET_PROTECT_REQ_MSG
```

27.3.6.2 T_MC_SET_PROTECT RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      UINT32            card_status;
} T_MC_CLR_PROTECT_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -write protect group is invalid; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check the Card status

27.3.7 Clear write protect

The T_MC_CLR_PROTECT _REQ_MSG message can be used to clear the write protection of the addressed write protect group. This message is similar to the mc_clr_write_protect() function (see 27.2.7). The driver responds to an MC_CLR_PROTECT_REQ_MSG message with a T_MC_CLR_PROTECT_RSP_MSG message.

27.3.7.1 T_MC_CLR_PROTECT_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_RCA          rca;
      UINT32            wr_prot_group;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_CLR_PROTECT_REQ_MSG
```

27.3.7.2 T_MC_CLR_PROTECT_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      UINT32            card_status;
} T_MC_CLR_PROTECT_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -write protect group is invalid; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check the Card status

27.3.8 Get write protect

The T_MC_GET_PROTECT _REQ_MSG message can be used to read 32 write protection bits representing 32 write protect groups starting at a specified address. This message is similar to the mc_get__write_protect() function (see 27.2.8). The driver responds with a T_MC_GET_PROTECT_RSP_MSG message. The *wr_prot_grps* variable contains the write protection groups.

27.3.8.1 T_MC_GET_PROTECT_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_RCA          rca;
      UINT32            wr_prot_group;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_GET_PROTECT_REQ_MSG
```

27.3.8.2 T_MC_GET_PROTECT RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      UINT32            card_status;
      UINT32            wr_prot_grps;
} T_MC_GET_PROTECT_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -write protect group is invalid; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check the Card status

27.3.9 Get card status

The T_MC_CARD_STATUS_REQ_MSG message can be used to read 512-bit SD status register of an SD-card. This message is similar to the mc_sd_get_card_status() function (see 27.2.9). The driver responds with a T_MC_CARD_STATUS_RSP_MSG message.

27.3.9.1 T_MC_CARD_STATUS_REQ_MSG

```
typedef struct {
      T_RV_HDR              os_hdr;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_CARD_STATUS_REQ_MSG
```

27.3.9.2 T_MC_CARD_STATUS_RSP_MSG

```
typedef struct {
      T_RV_HDR              os_hdr;
      T_RV_RET              result;
      UINT32                card_status;
} T_MC_CARD_STATUS_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

27.3.10 Get SD card status

The T_MC_CARD_SD_STATUS_REQ_MSG message can be used to read -bit status register of an MMC/SD-card. This message is similar to the mc_get_card_status() function (see27.2.21). The driver responds with a T_MC_SD_CARD_STATUS_RSP_MSG message.

27.3.10.1 T_MC_SD_CARD_STATUS_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      UINT8             *sd_status_p;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_SD_CARD_STATUS_REQ_MSG;
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

27.3.10.2 T_MC_SD_CARD_STATUS_RSP_MSG

```
typedef struct {
      T_RV_HDR    os_hdr;
      T_RV_RET    result;
      UINT8 *sd_status_p;
} T_MC_SD_CARD_STATUS_RSP_MSG;
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

27.3.11 Update acquisition

The T_MC_UPDATE_ACQ_REQ_MSG message can be used to start an identification cycle of the card stack. This message is similar to the mc_update_acq () function (see 27.2.11). The driver responds with a T_MC_UPDATE_ACQ_RSP_MSG message. The *stack_size* variable in the response message contains the number of connected MMC/SD-cards.

27.3.11.1 T_MC_UPDATE_ACQ_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_UPDATE_ACQ_REQ_MSG
```

27.3.11.2 T_MC_UPDATE_ACQ_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      UINT16            stack_size;
```

```
} T_MC_UPDATE_ACQ_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_INVALID_PARAMETER	-invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

27.3.12 Reset cards

This function resets all cards to idle state. This message is similar to the mc_reset() function (see 27.2.12). The driver responds with a T_MC_RESET_MSG message.

27.3.12.1 T_MC_RESET_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_RESET_REQ_MSG
```

27.3.12.2 T_MC_RESET_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
} T_MC_RESET_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

27.3.13 T_MC_READ_OCR_REQ_MSG

The T_MC_READ_OCR_REQ_MSG message can be used to retrieve the 32-bit OCR register of an MMC/SD-card. This message is similar to the mc_read_OCR () function (see 27.2.15). The driver responds with a T_MC_READ_OCR_RSP_MSG message.

27.3.13.1 T_MC_READ_OCR_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_RCA          rca;
      UINT32            *ocr_p;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_READ_OCR_REQ_MSG
```

27.3.13.2 T_MC_READ_OCR_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      UINT32            *ocr_p;
} T_MC_READ_OCR_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success

RV_NOT_READY	The driver is in “DETACHED” state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -ocr_p is NULL; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

27.3.14 Read CID

The T_MC_READ_CID_REQ_MSG message can be used to retrieve the 128-bit OCR register of an MMC/SD-card. This message is similar to the mc_read_CID () function (see 27.2.16). The driver responds with a T_MC_READ_CID_RSP_MSG message.

27.3.14.1 T_MC_READ_CID_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_RCA          rca;
      UINT8             *cid_p;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_READ_CID_REQ_MSG
```

27.3.14.2 T_MC_READ_CID_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      UINT8       *cid_p;
} T_MC_READ_CID_RSP_MSG
```

The possible values for ‘result’ are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in “DETACHED” state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -cid_p is NULL; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

27.3.15 Read CSD

The T_MC_READ_CSD_REQ_MSG message can be used to retrieve the 128-bit CSD register of an MMC/SD-card. This message is similar to the mc_read_CSD() function (see 27.2.17). The driver responds with a T_MC_READ_CSD_RSP_MSG message.

27.3.15.1 T_MC_READ_CSD_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_RCA          rca;
      UINT8             *csd_p;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_READ_CSD_REQ_MSG
```

27.3.15.2 T_MC_READ_CSD_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
```

```
      UINT8             *csd_p;
} T_MC_READ_CSD_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -csd_p is NULL; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

27.3.16 Write CSD field

The T_MC_WRITE_CSD_REQ_MSG message can be used to write the programmable part of the -bit CSD register of an MMC/SD-card. This message is similar to the mc_write_CSD() function (see **Error! Reference source not found.**). The driver responds with a T_MC_WRITE_CSD_RSP_MSG message.

27.3.16.1 T_MC_WRITE_CSD_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_RCA          rca;
      T_MC_CSD_FIELD    field;
      UINT8             value;
      T_MC_SUBSCRIBER   subscriber;
} T_MC_WRITE_CSD_REQ_MSG
```

27.3.16.2 T_MC_WRITE_CSD_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      UINT32            card_status;
} T_MC_WRITE_CSD_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -invalid field; -invalid value; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful. Check the card status.

27.3.17 Read scr

The T_MC_SD_READ_SCR_REQ_MSG message can be used to retrieve the 64-bit SCR register of an SD card. This message is similar to the mc_sd_read_SCR () function (see 27.2.18). The driver responds with a T_MC_SD_READ_SCR_RSP_MSG message.

27.3.17.1 T_MC_SD_READ_SCR_REQ_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_MC_RCA          rca;
      UINT8             *scr_p;
```

```
      T_MC_SUBSCRIBER   subscriber;
} T_MC_SD_READ_SCR_REQ_MSG
```

27.3.17.2 T_MC_SD_READ_SCR_RSP_MSG

```
typedef struct {
      T_RV_HDR          os_hdr;
      T_RV_RET          result = <RV_OK | RV_INVALID_PARAMETER |
                                  RV_NOT_READY | RV_INTERNAL_ERROR>
      UINT8             *scr_p;
} T_MC_SD_READ_SCR_RSP_MSG
```

The possible values for 'result' are:

Id	Definition
RV_OK	Success
RV_NOT_READY	The driver is in "DETACHED" state. No card is found
RV_INVALID_PARAMETER	-rca is invalid; -scr_p is NULL; -invalid subscriber;
RV_INTERNAL_ERROR	There was an internal error while executing the request. Execution was unsuccessful.

27.4 Types definition

27.4.1 T_MC_CSD_FIELD

```
typedef enum {
      CSD_FIELD_FILE_FORMAT_GRP,
      CSD_FIELD_COPY,
      CSD_FIELD_PERM_WRITE_PROTECT,
      CSD_FIELD_TMP_WRITE_PROTECT,
      CSD_FIELD_FILE_FORMAT,
      CSD_FIELD_FILE_ECC,
      CSD_FIELD_FILE_CRC,
} T_MC_CSD_FIELD;
```

27.4.2 T_MC_CSD_ACTION

```
typedef enum {
      CSD_ACTION_WRITE,
      CSD_ACTION_ERASE,
} T_MC_CSD_ACTION;
```

27.4.3 T_MC_SUBSCRIBER

```
typedef UINT16 T_MC_SUBSCRIBER;
```

27.4.4 T_MC_DMA_MODE

```
typedef enum {
     MC_FORCE_CPU,
     MC_FORCE_DMA,
     MC_DMA_AUTO
} T_MC_DMA_MODE;
```

27.4.5 T_MC_RW_MODE

```
typedef enum {
    MC_RW_STREAM,
    MC_RW_BLOCK
} T_MC_RW_MODE;
```

Appendices

B. Acronyms

API	Application Programming Interface
CAMA	Camera Application
CAMD	Camera driver
DMA	Direct memory access
EMIF	External Memory Interface
FAT	File Allocation Table
FFS	Flash file system
FIFO	First in first out
I/O	Input/Output
ICT	Company name: Industriële Computer Toepassingen
IMG	Image library
JPEG	Joint Photographer Engineering Group (compression algorithm)
LFS	Linear File System
MCU	Micro-Controller Unit
MIDI	
MMC	MultiMedia Card
MMU	Memory management unit
NOR	
POSIX	Portable Operating System Interface
QCIF	Quarter Common Intermediate Format: resolution is 144 by 176
R2D	Riviera 2D Graphics Library
RFS	Riviera File System
RGB	Colorspace: Red, Green, Blue components of color.
SD	Secure Digital
SWE	software Entity
TI	Company name: Texas Instruments
VGA	video graphics array: resolution is 640 by 480
YUV	Colorspace: Y stands for the luminance component (the brightness) and U and V are the chrominance (color) components.

C. References

[1]. TCS3.1 Documentation on R2D – “88_02_03_00723_RIV151_Riviera2D_api.doc”
[2]. Datasheet of Hitachi HD66772 – “HD66772 - LCD Source Driver - 2002.10.pdf”
[3]. Datasheet of Philips LPH8754 – “LPH8754-1 - 2.2 inch 176x220 TFT - 2003.05.12.pdf”
[4]. Datasheet of Sitronix ST7541 – “ST7541_4_gray_scale_dot_matrix_LCD_controller_driver_v1.1.pdf”

D. Agilent ADCM-2700 camera capabilities

```
struct T_CAMA_CAMERA_CAPABILITIES
  {
  UINT16 number_of_resolutions = 1;
struct
    {
      T_CAMA_RESOLUTION resolution = CAMA_VGA;
      T_CAMA_ENCODING encoding = CAMA_YUV_INTERLEAVED;
      UINT16 width = 640;
      UINT16 height = 480;
      UINT8 R_bits = *;
      UINT8 G_bits = *;
      UINT8 B_bits = *;
      UINT8 Y_bits = 8;
      UINT8 U_bits = 4;
      UINT8 V_bits = 4;
      UINT16 max_zoom = 0; /* no zoom */
    } resolution[];
  BOOL black_and_white = FALSE; /* not supported */
  BOOL flip_x = FALSE; /* not supported */
  BOOL flip_y = FALSE; /* not supported */
  BOOL rotate = FALSE; /* not supported */
  BOOL positioning = FALSE; /* not supported */
  BOOL refresh_rate = FALSE; /* not supported (i.e. is fixed) */
} cama_camera_capabilities
```

*=not applicable

E. Glossary

Object	An RFS object is an ordinary flat file or a directory. Objects in RFS are hierarchically organised in directories and sub-directories. Each object must have a unique name within the directory it resides. Whether the object names are case sensitive depends on the file system (most file systems are case sensitive, but for example FAT is not case sensitive). An object name is valid if it contains a combination of the following characters: a-z, A-Z, 0-9, %, $, #, ‘.’ (dot), ‘+’ (plus), ‘-‘ (dash), ‘_’ (underscore) and ‘,’ (comma).
Directory	An object in RFS is a directory. In directories, objects are hierarchically organised. These organised objects can be files or other directories.
Mount point	A mount point is an object in the file system. It is to be seen as a directory associated to a device partition of supported type.
Pathname	A pathname is defined as a mount point, a series of directory names separated by slashes (‘/’) and ending with a object name (file or directory). A pathname must begin with a slash and must never end with a slash. Example:

rfs_open("/mmc/programs/files/textfile.txt", RFS_O_CREATE);

- In this example “/mmc/programs/files/textfile.txt” is the path name
- ‘mmc’ is a mount point
- ‘programs’ is a directory
- ‘files’ is a directory
- ‘textfile.txt’ is a file(-name)

A pathname also can consist of only an object name (file). In this case the default mount point and directory rules are applicable.

Device

A device is the abstraction of any mass storage media or a partition of this media if it is a partitionable one.

Inode

An inode is identification for a group of data containing information about an object, organised for fast and efficient access.
Symbolic links work more or less the same as POSIX symbolic links except that they can only be used for referencing files and not other symbolic links or directories. In general, use of symbolic links is discouraged because they consume inodes and increase the object lookup time.