



---

Technical Document

## **GSM PROTOCOL STACK**

### **GPF**

## **OSX – CUSTOMER FRAME INTERFACE**

## **FUNCTIONAL INTERFACE DESCRIPTION**

---

Document Number:	06-03-10-ISP-0004
Version:	0.5
Status:	Draft
Approval Authority:	
Creation Date:	2000-Oct-25
Last changed:	2015-Mar-08 by MP
File Name:	osx_api.doc

## 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 acknowledgment. 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 minimize 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 recognized standards and that certain third parties may claim intellectual property rights therein. The supply of products and/or the licensing of software does 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.

No part of this document may be reproduced or transmitted in any form or by any means, electronically or mechanically, including photocopying and recording, for any purpose without the express written permission of TI.

## Change History

Date	Changed by	Approved by	Version	Status	Notes
2000-Oct-25	MP et al.		0.1	Being Processed	1
2001-May-04	MP et al.		0.2	Being Processed	2
2001-Sep-07	MP et al.		0.3	Submitted	3
2003-May-20	XINTE GRA		0.4	Draft	
2003-Sep-08	MP		0.5	Submitted	4

**Notes:**

1. Initial version
2. Function names changed
3. Document number changed
4. new API function `osx_config()`  
    `xSignalHeaderRec` modified

## Table of Contents

<b>Functional Interface Description .....</b>	<b>1</b>
1.1 Abbreviations .....	5
<b>2 Frame/Body Concept .....</b>	<b>6</b>
<b>3 Customer Frame Interface .....</b>	<b>7</b>
3.1 Data Types .....	7
3.1.1 T_VOID_STRUCT .....	7
3.1.2 T_ENUM_OS_QUEUE .....	7
3.1.3 xSignalHeaderRec .....	8
3.2 Constants .....	8
3.2.1 Return Codes .....	8
3.3 int_osx Functions .....	9
3.3.1 int_osx_alloc_prim() .....	9
3.3.2 int_osx_alloc_mem() .....	10
3.3.3 int_osx_free_prim() .....	11
3.3.4 int_osx_free_mem() .....	12
3.3.5 int_osx_send_prim() .....	13
3.3.6 int_osx_receive_prim() .....	14
3.3.7 int_osx_send_sig() .....	15
3.4 osx Functions.....	16
3.4.1 _osx_init.....	16
3.4.2 _osx_config() .....	17
3.4.3 _osx_open() .....	18
3.4.4 osx_alloc_prim() .....	19
3.4.5 osx_alloc_mem() .....	20
3.4.6 osx_free_prim().....	21
3.4.7 osx_free_mem().....	22
3.4.8 osx_send_prim() .....	23
3.4.9 osx_receive_prim().....	24
3.4.10 osx_send_sig() .....	25
<b>Appendices.....</b>	<b>26</b>
A. Acronyms .....	26
B. Glossary .....	26

## List of Figures and Tables

## List of References

- |                        |   |
|------------------------|---|
| <b>[ISO 9000:2000]</b> | International Organization for Standardization. Quality management systems - Fundamentals and vocabulary. December 2000 |
|------------------------|---|

## 1.1 Abbreviations

RTOS	Real-time Operating System
VSI	Virtual System Interface
PEI	Protocol Stack Entity Interface

## 2 Frame/Body Concept

The frame body concept has been designed in the context of the G23 Protocol Stack. In the case of the G23 Protocol Stack, a process represents the protocol logic of a protocol stack entity. This architecture separates the process functionality into two logical modules, the process frame and the process body. Common process functionality is located in the process frame. The main process functionality is located in the process body.

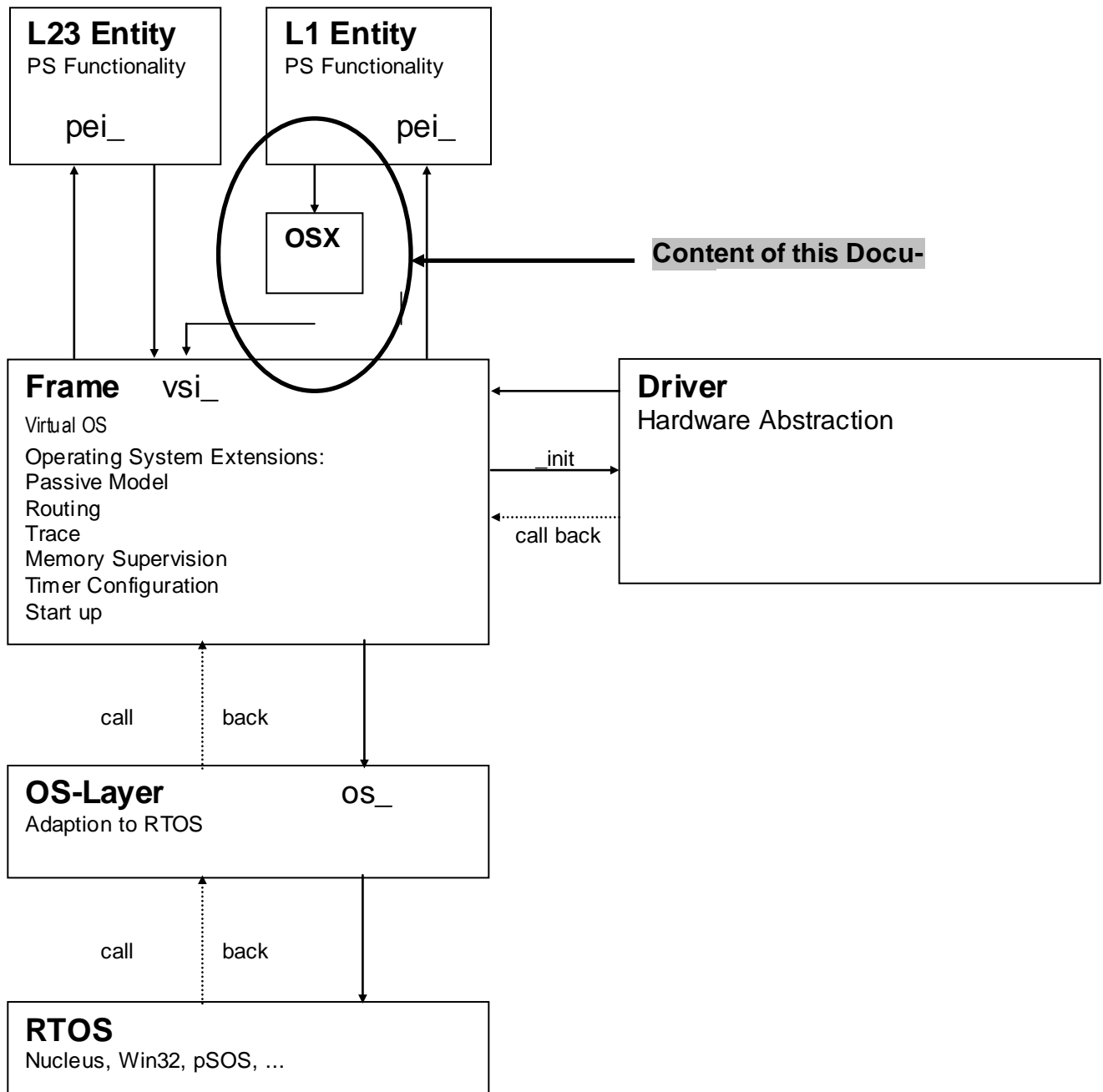


Figure 1: Structure of GSM Protocol Stack

The process frame has two interfaces. The Virtual System Interface (VSI) is the frames functional interface to be accessed by the bodies. The Operating System Interface (OS) is also a functional interface and provides the interface to the Real-time Operating System (RTOS). This interface is encapsulated in the "OS Layer" in order to keep the frame itself independent from the underlying RTOS.

In the OS Layer, the request of system resources by the protocol stack entities via the VSI is adapted to the implemented RTOS.

The intention of this interface is to provide a set of function calls that is independent of the underlying RTOS. If an RTOS does not supply all the features described in the following, e.g. the possibility of periodic timers, this must be adapted within the OS Interface.

Some functionality of the interface described in the following may not be necessary for all RTOSs and therefore some functions do not have to be filled with code. The releasing of queue handles if they are not longer used might not always be necessary when communication is closed.

The OSX interface has been designed for customers who do not like to access the Condat frame directly in order to keep there software independent of the Condat VSI.

## 3 Customer Frame Interface

The customer frame interface is split up into two parts. There are functions beginning with `osx` and functions beginning with `int_osx`. The difference between the two variants is that the `osx` functions do not have a parameter *caller* and the access to queues is done via identifiers that are determined by the customer. The `int_osx` functions do also have a parameter *caller* that is needed for the memory supervision feature and they access the queues via a frame queue handles.

**For performance reasons it is recommended to use the `int_osx` interface.**

The `int_osx` functions will be used by customers who use the queue handles that have been retrieved in the `pei_init()` function of the specific task to access the interface, refer to 3.3.

Also there are customers who do not really use Condat's PEI interface because they run their task in active body variant. These customers also have a PEI interface but it is provided by Condat and consists only of a `pei_create()`, `pei_init()` and `pei_run()` function. In this case Condat need to know which communication channels are required for this task and which identifiers the customer will use to access them. For this reason the communication channels for the `osx` have to be opened by calling an `osx_open()` function for every channel to be used later to assign a queue handle to every identifier. Also this task has to be registered as the `osx` caller by a call of `osx_open()` with specific parameters, refer to 3.4.3. Before `osx_open()` is called the first time the function `osx_init()` has to be called. Due to this registration and opening of communication channels `osx` functions internally know their caller and are able to convert the passed queue identifiers into queue handles. For `osx` functions refer to 3.4.

### 3.1 Data Types

#### 3.1.1 T\_VOID\_STRUCT

**Definition:** `typedef unsigned long T_VOID_STRUCT`

**Description:** Pointers of type `T_VOID_STRUCT` are passed to functions in order to avoid warnings when using void pointers with a subsequent cast operation within the called function.

#### 3.1.2 T\_ENUM\_OS\_QUEUE

**Definition:**

`typedef enum`

```
{
    L1_QUEUE,          /* internal L1 communication */
    DL_QUEUE,          /* L1->DL */
}
```

```

RR_QUEUE,          /* L1->RR                      */
GRR_QUEUE,         /* L1->GRR                      */
LLC_QUEUE,         /* L1->LLC e.g. ciphering via CCI */
SNDCP_QUEUE,       /* L1->SNDCP e.g. compression via CCI */
MAX_OSX_QUEUE
} T_ENUM_OS_QUEUE;

```

**Description:** The values of this enum are used as queue\_types for the osx functions.

### 3.1.3 xSignalHeaderRec

**Definition:**

```

typedef struct xSignalHeaderStruct
{
    int      SignalCode;
    int      _dummy1;
    int      _dummy2;
    int      _dummy3;
    void     *SigP;
    int      _dummy4;
} xSignalHeaderRec;

```

**Description:** Pointers of this type are used as parameter for the (int\_)osx\_send\_prim and (int\_)osx\_receive\_prim functions.

## 3.2 Constants

### 3.2.1 Return Codes

OSX_OK	0	successful execution
OS_ERROR	-1	error



## 3.3 int\_osx Functions

### 3.3.1 int\_osx\_alloc\_prim()

#### Function definition:

xSignalHeaderRec\* int\_osx\_alloc\_prim ( short caller, unsigned long len )

#### Parameters:

Type	Name	Meaning	
short	caller	handle of calling task	IN
unsigned long	len	size of partition to be allocated	IN

#### Return:

Type	Meaning
xSignalHeaderRec*	pointer to allocated partition

#### Description:

The function int\_osx\_alloc\_prim() allocates a partition of a size that is sufficient to hold *len* bytes.

If no free partition is available at calling time the calling task is suspended. If the request has been satisfied but the underlying OS-layer function had to wait for a free partition a corresponding message

"SYSTEM WARNING: Waited for partition, task *task*, size *size*, opc *opc*" is traced.

If the caller is a non-task thread, the function returns immediately regardless whether or not the request can be satisfied. In this case the calling task is suspended forever and an error message

"SYSTEM ERROR: No Partition available, task *task* size *size*, opc *opc* " is traced and the system is stopped.

In both cases the complete partition pool of the requested size is dumped via the test interface.

### 3.3.2 int\_osx\_alloc\_mem()

#### Function definition:

void \* int\_osx\_alloc\_mem ( short caller, unsigned long len )

#### Parameters:

Type	Name	Meaning	
short	caller	handle of calling task	IN
unsigned long	len	size of memory buffer to be allocated	IN

#### Return:

Type	Meaning
void *	pointer to allocated memory buffer

#### Description:

The function int\_osx\_alloc\_mem() allocates a memory buffer of a size that is sufficient to hold *len* bytes.

If no free partition is available at calling time the calling task is suspended. If the request has been satisfied but the underlying OS-layer function had to wait for a free partition a corresponding message

“SYSTEM WARNING: Waited for memory, task *task*, size *size*” is traced.

If the caller is a non-task thread, the function returns immediately regardless whether or not the request can be satisfied. If no partition is available the message

“SYSTEM ERROR: No memory available, task *task* size *size*” is traced and the system is stopped.

In both cases the complete partition pool of the requested size is dumped via the test interface.

### 3.3.3 int\_osx\_free\_prim()

#### Function definition:

void osx\_free\_prim ( short caller, xSignalHeaderRec \*prim\_ptr )

#### Parameters:

Type	Name	Meaning	
short	caller	handle of calling task	IN
xSignalHeaderRec *	prim_ptr	pointer to partition to be freed	IN

**Return:** ---

#### Description:

The function int\_osx\_free\_prim() deallocates the memory partition to which prim\_ptr points.

### 3.3.4 int\_osx\_free\_mem()

#### Function definition:

void int\_osx\_free\_mem ( void \* mem\_ptr )

#### Parameters:

Type	Name	Meaning	
short	caller	handle of calling task	IN
void *	mem_ptr	pointer to memory to be freed	IN

**Return:** ---

#### Description:

The function int\_osx\_free\_prim() deallocates the memory partition to which prim\_ptr points.

### 3.3.5 int\_osx\_send\_prim()

#### Function definition:

```
void int_osx_send_prim ( short caller, xSignalHeaderRec *prim_ptr, short queue_handle )
```

#### Parameters:

Type	Name	Meaning	
short	caller	handle of calling task	IN
xSignalHeaderRec *	prim_ptr	pointer to message to be sent	IN
short	queue_handle	destination queue handle	IN

**Return:** ---

#### Description:

The function int\_osx\_send\_prim() sends the message to which *prim\_ptr* points to the message queue identified by *queue\_handle*.

If there is no space available in the destination queue at calling time the calling task is suspended. If the request has been satisfied but the underlying OS-layer function had to wait for a space in the destination queue a corresponding message

"SYSTEM WARNING: *task* waited for space in *task* queue" is traced.

If the caller is a non-task thread, the function returns immediately regardless whether or not the request can be satisfied. If no space in the destination queue is available the message

"SYSTEM ERROR: *task* write attempt to *task* queue failed" is traced.

### 3.3.6 int\_osx\_receive\_prim()

#### Function definition:

xSignalHeaderRec \* int\_osx\_receive\_prim ( short caller, short queue\_handle )

#### Parameters:

Type	Name	Meaning	
short	caller	handle of calling task	IN
short	queue_handle	queue handle	IN

#### Return:

Type	Meaning
xSignalHeaderRec	pointer to received message

#### Description:

The function int\_osx\_receive\_prim() supervises the message queue specified by the parameter *queue\_handle*.

The calling task is suspended until a message is available in the queue.

### 3.3.7 int\_osx\_send\_sig()

#### Function definition:

```
void int_osx_send_sig ( short caller, unsigned long opc, void *signal_ptr, short queue_handle )
```

#### Parameters:

Type	Name	Meaning	
short	caller	handle of calling task	IN
unsigned long	opc	operation code of signal	IN
void *	signal_ptr	pointer to signal to be sent	IN
short	queue_handle	destination queue handle	IN

**Return:** ---

#### Description:

The function `int_osx_send_sig()` sends the message to which *signal\_ptr* points to the message queue identified by *queue\_handle*. Messages sent as a signal have a higher priority than primitives when the destination queue is read out.

If there is no space available in the destination queue at calling time the calling task is suspended. If the request has been satisfied but the underlying OS-layer function had to wait for a space in the destination queue a corresponding message

"SYSTEM WARNING: *task* waited for space in *task* queue" is traced.

If the caller is a non-task thread, the function returns immediately regardless whether or not the request can be satisfied. If no space in the destination queue is available the message

"SYSTEM ERROR: *task* write attempt to *task* queue failed" is traced.

## 3.4 osx Functions

### 3.4.1 \_osx\_init

**Function definition:**

```
void _osx_init ( void )
```

**Parameters:** ---**Return:** ---**Description:**

The function \_osx\_init() initializes the table that is used for the assignment of queue\_types to queue\_handles that is done in osx\_open(), refer to 3.4.3.

This function must only be called by processes that intend to access the queues via queue\_types instead of the handles retrieved in the pei\_init() function.



### 3.4.2 \_osx\_config()

#### Function definition:

short \_osx\_config ( const char \* config )

#### Parameters:

Type	Name	Meaning	
char *	config	configuration string	IN

#### Return:

Type	Meaning	
short	OSX_OK	success
	OSX_ERROR	error

#### Description:

The function \_osx\_config() allows dynamic configuration of the OSX layer. Currently only the disabling of L1S traces is implemented.

L1S traces are sent from L1S to L1A via primitives using the primitive id TRACE\_INFO (0x7d). When L1S traces are dynamically disabled these primitives are discarded during a osx\_send\_prim() command and the memory used for the primitive is freed.

To disabled L1S traces send a configuration primitive "CONFIG L1S\_TRACE\_DISABLE" to the L1 entity. To enable the traces send primitive "CONFIG L1S\_TRACE\_ENABLE" to L1.

### 3.4.3 \_osx\_open()

#### Function definition:

short \_osx\_open ( short caller, unsigned short queue\_type, short queue\_handle )

#### Parameters:

Type	Name	Meaning	
short	caller	task handle of the caller	IN
unsigned short	queue_type	index of a queue	IN
short	queue_handle	handle of a queue	IN

#### Return:

Type	Meaning	
int	OSX_OK	success
	OSX_ERROR	error

#### Description:

The function \_osx\_open() assigns a queue\_type to a queue\_handle. As a consequence of this assignment the queue\_type is converted to a queue\_handle in the following queue accesses via osx\_send\_prim() and osx\_receive\_prim().

The queue\_handles are the handles returned by vsi\_c\_open() calls in the pei\_init() function of the xxx\_pei.c module.

This function must only be called by processes that intend to access the queues via queue\_types instead of the handles retrieved in the pei\_init() function.

### 3.4.4 osx\_alloc\_prim()

#### Function definition:

xSignalHeaderRec\* osx\_alloc\_prim ( unsigned long len)

#### Parameters:

Type	Name	Meaning	
unsigned long	len	size of partition to be allocated	IN

#### Return:

Type	Meaning
xSignalHeaderRec*	pointer to allocated partition

#### Description:

The function osx\_alloc\_prim() retrieves the handle of the caller and calls the function int\_osx\_alloc\_prim() to allocate a partition of a size that is sufficient to hold *len* bytes.

If no free partition is available at calling time the calling task is suspended. If the request has been satisfied but the underlying OS-layer function had to wait for a free partition a corresponding message

"SYSTEM WARNING: Waited for memory, task *task*, size *size*" is traced.

If the caller is a non-task thread, the function returns immediately regardless whether or not the request can be satisfied. If no partition is available the message

"SYSTEM ERROR: No memory available, task *task* size *size*, opc *opc* " is traced and the system is stopped.

In both cases the complete partition pool of the requested size is dumped via the test interface.

This function must only be called by processes that have registered themselves to access the osx in the pei\_init() function. The caller that is needed internally for memory supervision functionality is set to the caller of osx\_open() when called with queue\_type and queue\_handle set to zero.

### 3.4.5 osx\_alloc\_mem()

#### Function definition:

```
void* osx_alloc_mem ( unsigned long len)
```

#### Parameters:

Type	Name	Meaning	
unsigned long	len	size of memory buffer to be allocated	IN

#### Return:

Type	Meaning
void *	pointer to allocated memory

#### Description:

The function `osx_alloc_mem()` retrieves the handle of the caller and calls the function `int_osx_alloc_mem()` to allocate a memory buffer of a size that is sufficient to hold *len* bytes.

If no free partition is available at calling time the calling task is suspended. If the request has been satisfied but the underlying OS-layer function had to wait for a free partition a corresponding message

“SYSTEM WARNING: Waited for memory, task *task*, size *size*” is traced.

If the caller is a non-task thread, the function returns immediately regardless whether or not the request can be satisfied. If no partition is available the message

“SYSTEM ERROR: No memory available, task *task* size *size*” is traced and the system is stopped.

In both cases the complete partition pool of the requested size is dumped via the test interface.

This function must only be called by processes that have registered themselves to access the `osx` in the `pei_init()` function. The caller that is needed internally for memory supervision functionality is set to the caller of `osx_open()` when called with `queue_type` and `queue_handle` set to zero.

### 3.4.6 osx\_free\_prim()

#### Function definition:

```
void osx_free_prim ( xSignalHeaderRec *prim_ptr )
```

#### Parameters:

Type	Name	Meaning	
xSignalHeaderRec *	prim_ptr	pointer to partition to be freed	IN

**Return:** ---

#### Description:

The function `osx_free_prim()` retrieves the handle of the caller and calls the function `int_osx_free_mem()` to deallocate the memory to which `prim_ptr` points.

This function must only be called by processes that have registered themselves to access the `osx` in the `pei_init()` function. The caller that is needed internally for memory supervision functionality is set to the caller of `osx_open()` when called with `queue_type` and `queue_handle` set to zero.

### 3.4.7 osx\_free\_mem()

#### Function definition:

```
void osx_free_mem ( void * mem_ptr )
```

#### Parameters:

Type	Name	Meaning	
void *	mem_ptr	pointer to memory to be freed	IN

**Return:** ---

#### Description:

The function `_osx_free_mem()` retrieves the handle of the caller and calls the function `int_osx_free()` to deallocates the memory to which `mem_ptr` points.

This function must only be called by processes that have registered themselves to access the osx in the `pei_init()` function. The caller that is needed internally for memory supervision functionality is set to the caller of `osx_open()` when called with `queue_type` and `queue_handle` set to zero.

### 3.4.8 osx\_send\_prim()

#### Function definition:

```
void osx_send_prim ( xSignalHeaderRec *prim_ptr, T_ENUM_OS_QUEUE queue_type )
```

#### Parameters:

Type	Name	Meaning	
xSignalHeaderRec *	prim_ptr	pointer to message to be sent	IN
T_ENUM_OS_QUEUE	queue_type	destination queue identifier	IN

**Return:** ---

#### Description:

The function `osx_send_prim()` retrieves the handle of the caller and calls the function `int_osx_send_prim()` to send the message to which *prim\_ptr* points to the message queue identified by *queue\_type*.

The parameter *queue\_type* is converted to a queue handle by evaluation of a registry where the assignment of queue identifiers (*queue\_type*) to queue handles is stored during previous `osx_open()` calls.

If there is no space available in the destination queue at calling time the calling task is suspended. If the request has been satisfied but the underlying OS-layer function had to wait for a space in the destination queue a corresponding message

"SYSTEM WARNING: *task* waited for space in *task* queue" is traced.

If the caller is a non-task thread, the function returns immediately regardless whether or not the request can be satisfied. If no space in the destination queue is available the message

"SYSTEM ERROR: *task* write attempt to *task* queue failed" is traced.

is traced.

This function must only be called by processes that have registered themselves to access the `osx` in the `pei_init()` function. The caller that is needed internally for memory supervision functionality is set to the caller of `osx_open()` when the *queue\_handle* is assigned to the *queue\_type*.

### 3.4.9 osx\_receive\_prim()

#### Function definition:

xSignalHeaderRec \*osx\_receive\_prim ( T\_ENUM\_OS\_QUEUE queue\_type )

#### Parameters:

Type	Name	Meaning	
T_ENUM_OS_QUEUE	queue_type	queue identifier	IN

#### Return:

Type	Meaning
xSignalHeaderRec	pointer to received message

#### Description:

The function `osx_receive_prim()` retrieves the handle of the caller and calls the function `int_osx_receive_prim()` to supervise the message queue specified by the parameter `queue_type`.

The parameter `queue_type` is converted to a queue handle by evaluation of a registry where the assignment of queue identifiers (`queue_type`) to queue handles is stored during previous `osx_open()` calls.

The calling task is suspended until a message is available in the queue.

This function must only be called by processes that have registered themselves to access the osx in the `pei_init()` function. The caller that is needed internally for memory supervision functionality is set to the caller of `osx_open()` when the `queue_handle` is assigned to the `queue_type`.



### 3.4.10 osx\_send\_sig()

#### Function definition:

void osx\_send\_sig ( unsigned long opc, void \*signal\_ptr, T\_ENUM\_OS\_QUEUE queue\_type)

#### Parameters:

Type	Name	Meaning	
unsigned long	opc	operation code of signal	IN
void *	signal_ptr	pointer to signal to be sent	IN
T_ENUM_OS_QUEUE	queue_type	destination queue identifier	IN

**Return:** ---

#### Description:

The function osx\_send\_sig() retrieves the handle of the caller and calls the function int\_osx\_send\_sig() to send the message to which *signal\_ptr* points as signal to the message queue identified by *queue\_type*. Messages sent as a signal have a higher priority than primitives when the destination queue is read out.

The parameter *queue\_type* is converted to a queue handle by evaluation of a registry where the assignment of queue identifiers (*queue\_type*) to queue handles is stored during previous osx\_open() calls.

If there is no space available in the destination queue at calling time the calling task is suspended. If the request has been satisfied but the underlying OS-layer function had to wait for a space in the destination queue a corresponding message

"SYSTEM WARNING: *task* waited for space in *task* queue" is traced.

If the caller is a non-task thread, the function returns immediately regardless whether or not the request can be satisfied. If no space in the destination queue is available the message

"SYSTEM ERROR: *task* write attempt to *task* queue failed" is traced.

is traced.

This function must only be called by processes that have registered themselves to access the osx in the pei\_init() function. The caller that is needed internally for memory supervision functionality is set to the caller of osx\_open() when the queue\_handle is assigned to the queue\_type.

## Appendices

### A. Acronyms

**DS-WCDMA** Direct Sequence/Spread Wideband Code Division Multiple Access

### B. Glossary

**International Mobile Telecommunication 2000 (IMT-2000/ITU-2000)** Formerly referred to as FPLMTS (Future Public Land-Mobile Telephone System), this is the ITU's specification/family of standards for 3G. This initiative provides a global infrastructure through both satellite and terrestrial systems, for fixed and mobile phone users. The family of standards is a framework comprising a mix/blend of systems providing global roaming. <URL: <http://www.imt-2000.org/>>