Draft

TSC2.1 MFW Functional Specification

Document Number:	0001
Version:	001
Status:	Draft
Approval Authority:	
Creation Date:	2003-04-25
Last changed:	2003-05-02 by Harry Letch

Legal Notice

Change History

Date	Changed by	Version	Status	Changes/Comments
25/Apr/2003	Harry Letch	001	Draft	Initial version

0 Table of Contents

0.1 References, Abbreviations, Terms

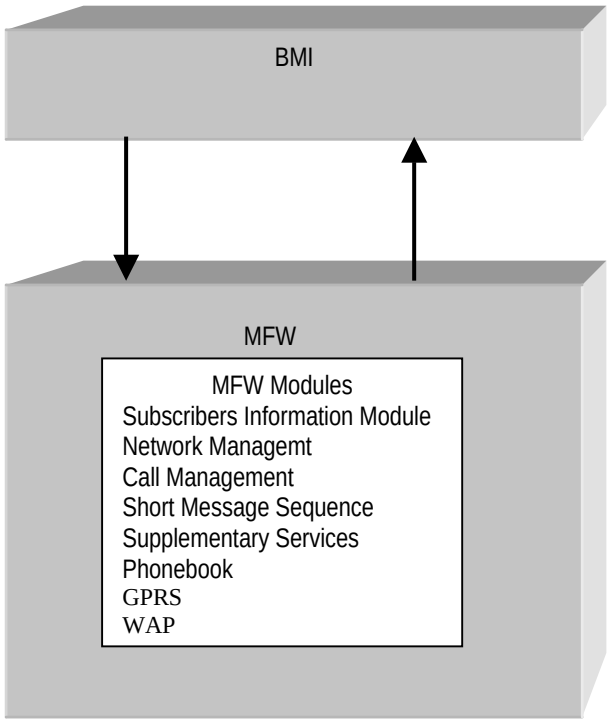
ACI	Application Controller Interface
ADN	Abbreviated Dialing Number
BDN	Barred Dialing Number
BMI	Basic MMI
CM	Call Management
CPMS	Common PCN Handset Specification
ECC	Emergency Call Number
FDN	Fixed Dialing Number
GSM	Global System for Mobile Communications
GPRS	General Packet Radio Services
LDN	Last Dialed Number
LMN	Last Missed Number
LRN	Last Received Number
MFW	Mobile Framework
MMI	Man Machine Interface
NM	Network Management
PHB	Phonebook
PLMN	Public Land Mobile Network
PUK	Personal Unlocking Key
SDN	Service Dialing Number
SAT	Sim Application Toolkit
SIM	Subscribers Information Module
SMS	Short Message Sequence
SS	Supplementary Service

1 Introduction

The Basic Man Machine Interface (BMI) of a mobile defines the user interface and uses the services of the the MMI Framework (MFW) and the Application Control Interface (ACI).

This document gives an overview of the BMI and MFW and details the interface between them. It gives a description of the function calls made by BMI to MFW and the Events returned to BMI by MFW.

The following diagram shows the inter-working between the BMI and the MFW framework component.



The BMI makes a call to the appropriate MFW module and receives an MFW event as a reply.

2 BMI Overview

2.1 Main BMI Modules

The BMI is divided into a number of modules which use common components in their presentation

2.1.1 Standard Modules

Phonebook
Call
Pins
Supplementary Services (SS)
Short Message Services (SMS)

2.1.2 Common Components

Menus
Editors
Languages
Dialogue
List
Icons

2.1.3 Additional Modules

Wireless Application Protocol

The features and behavioural characteristics of the BMI are described in detail in Ref nn.

3 MFW Overview

3.1 Main MFW Modules

The MFW is divided into a number of modules which use common components in their presentation.

3.1.1 Standard Modules

SIM
Phonebook
Call
Supplementary Services (SS)
Short Message Sequences (SMS)

3.1.2 Common Components

Window
Keypad
Timers
Menus
Editors
Icons

3.1.3 Additional Modules

Wireless Application Protocol (WAP)
General Packet Radio Resources (GPRS)
CPHS

3.2 MFW Standard & Additional Modules

The standard and additonal modules are detailed in the GSM Functional Description See Ref 5.

3.3 MFW Common Components

3.3.1 MFW Windows

An Mfw window is created at the beginning of a task and is reponsible for:

- Handling the visual direct output to the LCD if applicable.
- Container of events handlers for keyboard, timer, network, call, SIM, SAT and phonebook.
- Container for generic user data.

Following is a detailed list of the functions needed to implement a window.

3.3.1.1 Create

This creates a window and inserts it into the window tree. See Ref 6.1

Function :	winCreate	(hwin, attr, mask, handler)	
Parameters:	MfwHnd	hwin.	The parent window.
	MfwWinAttr	*attr	The attributes, mainly the position and the size.
	MfwEvt	mask.	Specifies whether the window is visible or not.
	MfwCb	handler	Function called by MFW every time window is **Shown** or **Updated**.

The window won't be visible until the function winShow is called.
The win parameter is ignored in stack ordering mode, when the current focused window will be the parent.

3.3.1.2 Show

This updates a window and all its child elements and makes it visible

Function : winShow (hwin)

Parameters: MfwHnd hwin The window to be shown.

The call back function , added as a parameter in winCreate(), will be executed.
The window won't be visible if a child window overlaps it.

3.3.1.3 Delete

This removes a window and its components from the window tree.

Function : winDelete (hwin)

Parameters: MfwHnd hwin The window to be deleted.

3.3.1.4 Hide

The window is not removed from the tree, but it is not visible any more.

Function : winHide (hwin)

Parameters: MfwHnd hwin The window to be hidden.

3.3.1.5 UnHide

The window is made visible again.

Function : winUnHide (hwin)

Parameters: MfwHnd hwin The window to be shown.

3.3.1.6 Focus

Give a window focus so it is the first to be called when event arrives

Function : winFocus (hwin)

Parameters: MfwHnd hwin The window to be called.

3.3.1.7 Clear

Clear the contents of the window

Function : winClear (hwin)

Parameters: MfwHnd hwin The window to be cleared.

3.3.1.8 Update

Updates and shows the window tree below

Function : winUpdate (win)

Parameters: MfwWin *win A pointer to the window to be cleared.

3.3.1.9 Window example

The following shows the creation of a window *win_hello_world* and the callback used by **Show** and **Update**.

```
mfwHnd win_hello_world(mfwHnd parent)
{
        mfwHnd win = winCreate(parent, 0,E_WIN_VISIBLE,(T_MFW_CB)hello_world_win_cb);
        winShow(win);
        return win;
}

T_MFW_CB hello_world_win_cb(MfwEvt e, MfwWin *w)
{
        switch (e)
        {
```

```
        case MfwWinVisible:
                dspl_clearAll();
                dspl_txtOut(10,10,0,"HELLO WORLD");
                break;
        default:
                return 0;
        }
        return 1;
}
```

3.3.2 Keypad Handling

Keypad handling supports the following keypad events:

- Key pressed.
- Key released
- Long press key.

Following is a detailed list of the functions needed to implement keypad handling.

3.3.2.1 Create

Creates a keyboard handler and links it to the window hWin with the selected keys. See Ref 6.1

Function :	kbdCreate	(hwin, keys, handler)	
Parameters:	MfwHnd	hwin	The window handling the keypad.
	MfwEvtevent	keys	The keys to be handled
	MfwCbHandler	handler	Function called by MFW when there is a key event for that window.

The parameter keys is defined using a bit map as follows.
KEY_0, KEY_1, …, KEY_STAR, KEY_HASH define particular keys.
KEY_ALL for every key to be handled.
KEY_MAKE for release key to be handled,
KEY_AUTO for auto press to be handled,
KEY_LONG for long press...

3.3.2.2 Delete

Deletes the keyboard handler.

Function :	kbdDelete	(hwin)	
Parameters:	MfwHnd	hwin	The window handling the keypad.

3.3.2.3 Timeout

Sets the value of some keypad timers, as for example the long press timer.

Function :	kbdTime	(tLong, tAuto, tRepeat)	
Parameters:	MfwHnd	tLong	setup long press timer.
		tAuto	setup auto repeat timer
		tRepeat	save repeat interval

3.3.2.4 All keys

Registers the "handler" to be invoked with every keyboard event. Useful for global keypad utilities, like key tones or power off detection.

Function :	kbdTime	(handler)	
Parameters:	MfwCb	handler	

The scheduler first delivers the key event to the focused window, if the event is not handled then it passes it down the window tree until it finds a window that handles the event. A keyboard handler can decide to consume or pass the key event.

3.3.2.5 keypad example

The following shows the creation of a keypad handler and the callback used

```
kbd = kbdCreate(parent_win,KEY_ALL|KEY_MAKE,example_kbd_cb)

T_MFW_CB example_kbd_cb (MfwEvt e, MfwKbd *k)
{
        if ( !(e & KEY_MAKE) )
        {
                /*processes a release key event */
                return 1;
        }
         switch (k->code)
        {
         case KCD_LEFT:
                /* do something */
                break;

        default:
                return 0;
        }
}
```

3.3.3 Timers

Following is a detailed list of the functions needed to implement timers.

3.3.3.1 Create

Creates a timer.

Function :	timCreate	(hwin, timeout, handler)	
Parameters:	MfwHnd	hwin	The window handling the menu.
	S32	timeout	The time in Ms
	MfwCb	handler	Callback to be called when timer matures

3.3.3.2 Set

Sets a running timer to be a particular value.

Function :	timSetTime	(hwin, time)	
Parameters:	MfwHnd	hwin	The window handling the timer.
	S32	time	The time in Ms

3.3.3.3 Start, Stop, Delete

Starts, stops it or deletes a timer.

Function :	timStart, timStop, timDelete	(hwin)	
Parameters:	MfwHnd	hwin	The window handling the timer.

3.3.4 Menus

Menus are responsible for implementing the following:

- Shows a list of items in a defined area. The items can point to submenus or to functions .
- Scrolling either circular or linear.
- Supports bitmaps.

Following is a detailed list of the functions needed to implement keypad handling.

3.3.4.1 Create

Creates a menu with attributes provided. See Ref 6.1

Function : mnuCreate (hwin, attr)

Parameters:	MfwHnd	hwin	The window handling the menu.
	MfwMnuAttr	*attr	The attributes.

The attributes define the position and area and whether the menu is paged or a list.
For paged there is only one item per window, usually with an icon.
For a list, a pointer to the items and number of items.

3.3.4.2 Delete

Delete a menu.

Function : mnuDelete (hwin)

Parameters:	MfwHnd	hwin	The window handling the menu.

3.3.4.3 Language

Indicates the language to be handled.

Function : mnuLang (hwin, hLang)

Parameters:	MfwHnd	hwin	The window handling the menu.
	MfwHnd	hLang	The language Structure

3.3.4.4 Show

Display the menu.

Function : mnuShow (hwin)

Parameters:	MfwHnd	hwin	The window handling the menu.

3.3.4.5 Hide

The menu is not removed, but it is not visible any more.

Function : mnuHide (hwin)

Parameters:	MfwHnd	hwin	The menu to be hidden.

3.3.4.6 UnHide

The manu is made visible again.

Function : mnuUnhide (hwin)

Parameters:	MfwHnd	hwin	The manu to be shown.

3.3.4.7 Scroll

Scrolls up and down.

Function : mnuUp(hwin)., mnuDown(hwin)

Parameters:	MfwHnd	hwin	The window handling the menu.

3.3.4.8 Select

Selects the current item. Submenu or function will be called.

Function : mnuSelect (hwin)

Parameters:	MfwHnd	hwin	The menu to be shown.

3.3.4.9 Done

Goes back to the root of the menu.

Function : mnuDone (hwin)

Parameters:	MfwHnd	hwin	The menu to be shown.

3.3.4.10 Back

Goes back one level or leaves the menu.

Function :	mnuEscape	(hwin)	
Parameters:	MfwHnd	hwin	The menu to be shown.

3.3.5 Editors

Editors are responsible for implementing the following:

- Editing, buffering and display of strings on a particular area of a window.
- Multi line, word wrapping, basic scrolling capabilities, insert mode.

In the write mode, an editor has to be linked to a window and to a keyboard handler and will be shown when its parent window is shown.

Following is a detailed list of the functions needed to implement editors.

3.3.5.1 Create

This creates an editor and and links it to the window *hwin* it with a window. See Ref 6.1

Function :	winCreate	(hwin, attr, events, handler)	
Parameters:	MfwHnd	hwin	The parent window.
	MfwEdtAttr	*attr	The attributes, mainly the position, size and editor buffer.
	MfwEvt	events.	edit events to be handled. (Visible, etc)
	MfwCb	handler	Function called by MFW every time window is **Shown** or **Updated**.

3.3.5.2 Delete

Deletes the editor.

Function :	edtDelete	(hwin)	
Parameters:	MfwHnd	hwin	The window handling the editor.

3.3.5.3 Reset

Resets the internal editor variables.

Function :	edtReset	(hwin)	
Parameters:	MfwHnd	hwin	The window handling the editor.

3.3.5.4 Hide

Sets the editor to invisible without updating the screen.

Function :	edtHide	(hwin)	
Parameters:	MfwHnd	hwin	The window handling the editor.

3.3.5.5 UnHide

Sets the editor to visible without updating the screen.

Function :	edtUnhide	(hwin)	
Parameters:	MfwHnd	hwin	The window handling the editor.

3.3.5.6 Add

Adds a new character to the editor.

Function :	edtChar	(hwin, key)	
Parameters:	MfwHnd	hwin	The window handling the editor.
	int	key	The character to be added

3.3.5.7 Clear

Clears the editor buffer and updates the display.

Function :	edtClear	(hwin)	
Parameters:	MfwHnd	hwin	The window handling the editor.

4 BMI – MFW Interface

4.1 Initialisation of MFW

On power up the BMI intialises the MFW modules in turn by calling the appropriate function as follows.

4.1.1.1 Sim

Initialise sim and registration handlers .

Function : sim_init (void)

Parameters:

4.1.1.2 Level Indicators

Battery level & signal strength handlers.

Function : mmeCreate (hwin,events,handler)

Parameters:			
	MfwHnd	hwin	Null in this case
	MfwEvt	events	The event bit map
	MfwCb	handler	Callback handler function

The events parameter can be any combination of the following
MfwMmeSignal
MfwMmeBattery
MfwMmeBaState.

4.1.1.3 Pin

Pin verification & change handlers.

Function : sim_create (hwin,events,handler)

Parameters:			
	MfwHnd	hwin	Null in this case
	MfwEvt	events	The event bit map
	MfwCb	handler	Callback handler function

The events parameter has the value E_SIM_ALL_SERVICES.

4.1.1.4 Network Registration

Pin verification & change handlers.

Function : nm_create (hwin,events,handler)

Parameters:			
	MfwHnd	hwin	Null in this case
	MfwEvt	events	The event bit map
	MfwCb	handler	Callback handler function

The events parameter has the value E_NM_ALL_SERVICES.

4.1.1.5 SMS

Sms setup handler.

Function : sms_create (hwin,events,handler)

Parameters:			
	MfwHnd	hwin	Null in this case
	MfwEvt	events	The event bit map
	MfwCb	handler	Callback handler function

The events parameter can be any combination of the following:
E_SMS_MO_AVAIL
E_SMS_CMD_AVAIL
E_SMS_SAVE_AVAIL
E_SMS_MT

E_SMS_MO
E_SMS_CB
E_SMS_CB_RECEIVED
E_SMS_MT_RECEIVED
E_SMS_STATUS
E_SMS_MEM
E_SMS_OK
E_SMS_ERR
E_SMS_BUSY
E_SMS_READY
E_SMS_MEM_FULL
E_SMS_MEM_FREE

4.1.1.6 SS

Ss setup handler.

Function :	ss_create	(hwin,events,handler)	
Parameters:	MfwHnd	hwin	Null in this case
	MfwEvt	events	The event bit map
	MfwCb	handler	Callback handler function

The events parameter has the value 0xFFFF.

4.1.1.7 Call Management

Call setup handler.

Function :	cm_create	(hwin,events,handler)	
Parameters:	MfwHnd	hwin	Null in this case
	MfwEvt	events	The event bit map
	MfwCb	handler	Callback handler function

The events parameter has the value E_CM_ALL_SERVICES.

4.1.1.8 SAT

Sim Application Toolkit handler.

Function :	sat_create	(hwin,events,handler)	
Parameters:	MfwHnd	hwin	Null in this case
	MfwEvt	events	The event bit map
	MfwCb	handler	Callback handler function

The events parameter can be any combination of the following depending upon the options supported:

MfwSatRefresh
MfwSatTextOut
MfwSatGetKey
MfwSatGetString
MfwSatPlayTone
MfwSatSetupMenu
MfwSatSelectItem
MfwSatSendSMS
MfwSatSendSS
MfwSatCall
MfwSatSendUSSD
MfwSatSetEvents
MfwSatCcRes
MfwSatCcAlert
MfwSatIdleText
MfwSatExecAT
MfwSatSendDTMF
MfwSatDataRefreshed

MfwSatLaunchBrowser
MfwSatSessionEnd

4.1.1.9 GPRS

GPRS handler.

Function :	gprs_create	(hwin,events,handler)	
Parameters:	MfwHnd	hwin	Null in this case
	MfwEvt	events	The event bit map
	MfwCb	handler	Callback handler function

The events parameter has the value E_MFW_GPRS_ALL_EVENTS.

4.2 MFW Events

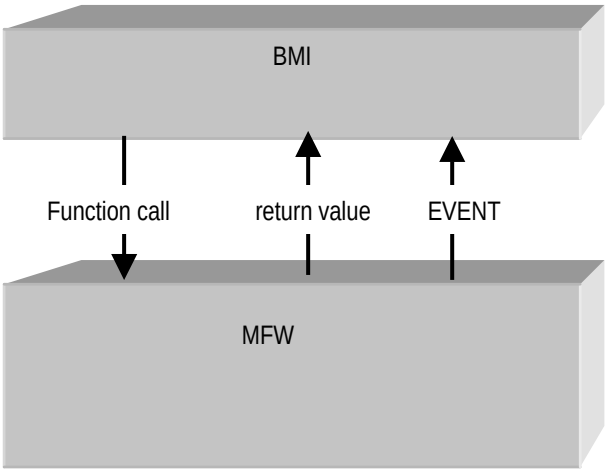
The following Mfw events can occur in the system:

- Keypad Events
- Equipment Events (battery strength, signal quality)
- Protocol stack Events
- Internal events (show / hide, update, delete, ...)

5 BMI - MFW Functional Description

5.1 Overview

The BMI starts a task by calling the appropriate MFW module function .

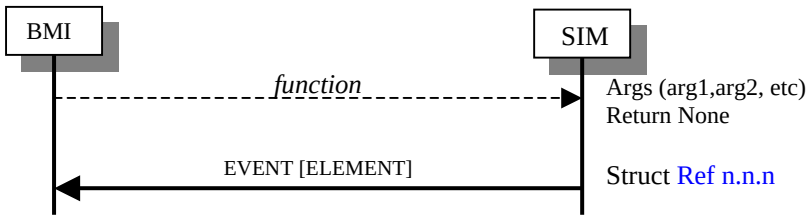


If the BMI expects an event to be returned then a pointer to a callback function is passed as one of the parameter of these calls. On completion of the task this callback is executed informing of the progress / status of the task in hand. The events can be simple enumerated types or else be associated with a Structure.

If the BMI is not expecting an event the return value from the function gives the progess / status of the task in hand.

5.2 Message Sequence Charts

This Section uses a form of message sequence chart to give a visual representation of task progress.



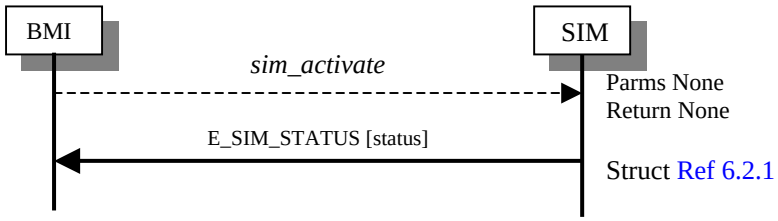
In the example above the BMI layer calls a *function* in the SIM module of the MFW. The Arguments passed are shown next to The function together with its return value.
This call results in an EVENT, which is sent to the BMI. Any Structure associated with the event is shown as a hyper link and the element of this Structure of interest is shown next to the event name.

5.3 Initialisation

When the user presses the key to switch the mobile on, the initialisation procedure is started. This procedure contains a welcome screen, initialisation of MFW and BMI modules, system configuration, for example audio setting, and creation of the necessary MFW event handler such as for keyboard, timer and network management, etc. After Initialisation the registration procedure of SIM activation, SIM lock check and network registration must be done.

5.4 SIM Activation

Once the initialisation is completed the BMI initiates the Sim Activation to ensure it is authentic. Callback routine used by BMI to receive MFW Events is *sim_event_cb_main.*

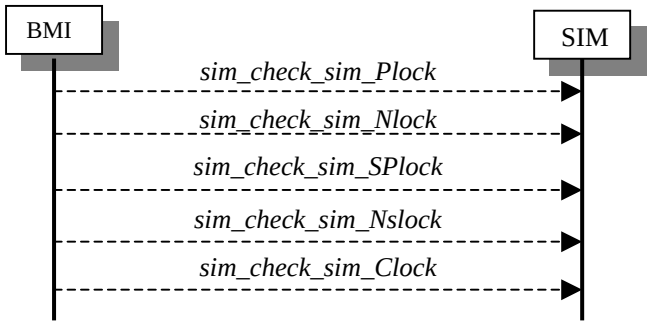


The function *sim_activate* is called is called to check the SIM authentication.
The event E_SIM_STATUS is received with parameter Structure T_MFW_SIM_STATUS.
The status value in this Structure can have the following values:
MFW_SIM_NO_PIN
MFW_SIM_PIN_REQ
MFW_SIM_PUK_REQ
MFW_SIM_INVALID_CARD
MFW_SIM_NO_SIM_CARD

The actions to be performed for each of the above cases, is described in detail in the sections below.

5.4.1 No Pin

PIN1 is disabled so the BMI starts is called to check the SIM lock. Five locks are defined in the GSM recommendation and each has to be checked in turn.



Any of The functions above can return one of the following values:
MFW_SIM_ENABLE
MFW_SIM_LOCKED
MFW_SIM_DISABLE
MFW_SIM_BLOCKED

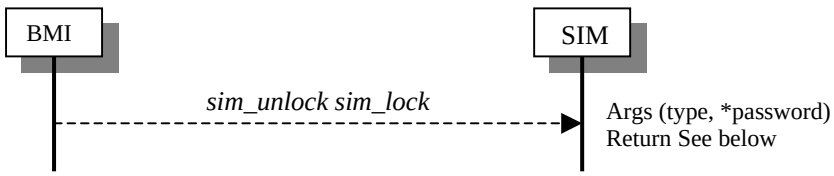
The actions to be performed for each of the above cases, is described in detail in the sections below.

5.4.1.1 MFW_SIM_ENABLE

The check has bee successful and the phone is unlocked.

5.4.1.2 MFW_SIM_LOCKED

The SIM card does not match the stored information so BMI creates the screen to enter the unlock password code



The function *sim_unlock_sim_lock* is called to verify the entered 'password'

The function returns one of the following:

MFW_SIM_*LOCK	unlock is successful (* is 'type' PLOCK, NLOCK, SPLOCK, NSLOCK or CLOCK)
MFW_SIM_UNLOCK_ERR	unlock has failed
MFW_SIM_BLOCKED	maximum number of attempts reached.

5.4.1.3 MFW_SIM_DISABLE

The function is disabled so the next check is performed.

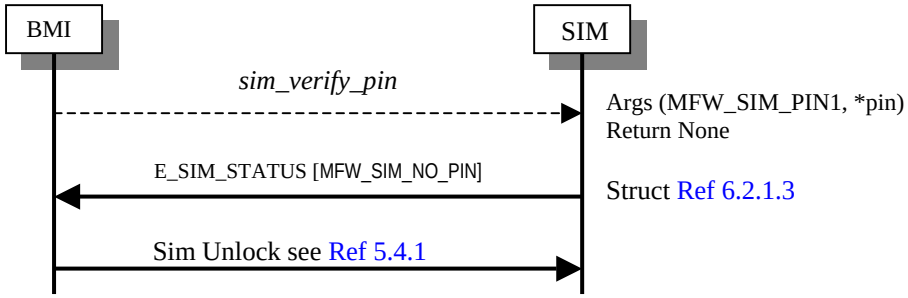
5.4.1.4 MFW_SIM_BLOCKED

The BMI can only start limited service.

5.4.2 Pin Required

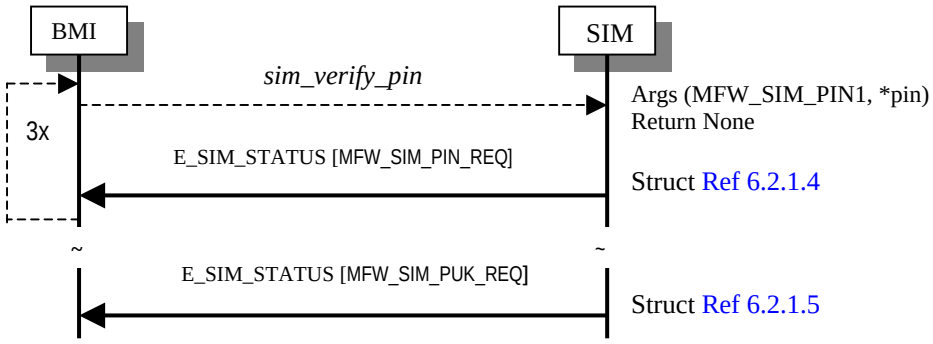
Limited Service is started, PIN1 is required so BMI creates editing screen to enter pin. The pin can either be correct or incorrect.

5.4.2.1 PIN1 correct



The function *sim_verify_pin* is called to verify the entered 'pin'.
The event E_SIM_STATUS with status MFW_SIM_NO_PIN is received indicating a correct PIN1 entered
BMI starts is called to check the pin lock. See Ref 6.2.1.3

5.4.2.2 PIN1 incorrect



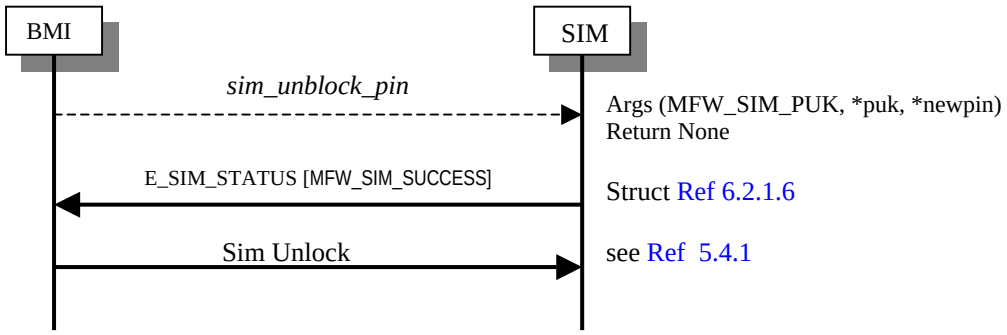
The function *sim_verify_pin* is called to verify the entered 'pin'.
The event E_SIM_STATUS with status MFW_SIM_PIN_REQ is received indicating an Incorrect PIN1 entered
BMI enters the pin and calls *sim_verify_pin* again
The event E_SIM_STATUS with status MFW_SIM_NO_PIN is received, indicating a correct PIN1 entered

Else if the number of retries has been reached
The event E_SIM_STATUS with status MFW_SIM_PUK_REQ is received indicating that the SIM card is blocked and the PUK must be entered.

5.4.3 Puk Required

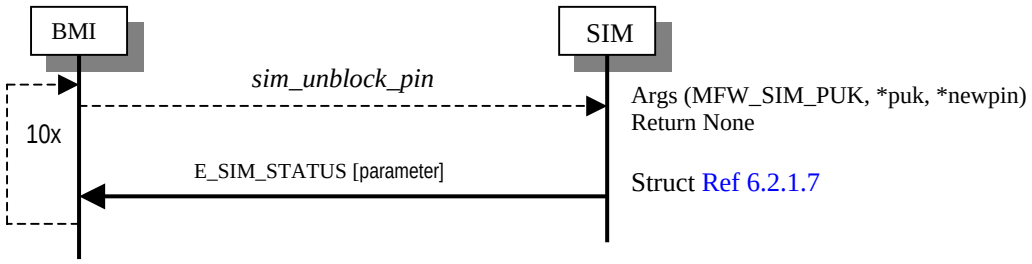
Limited service is started and as PUK1 is required BMI creates the editing screen to enter the puk. The puk can be either correct or incorrect.

5.4.3.1 PUK Correct



The function *sim_unblock_pin* is called to verify the entered 'puk'.
The event E_SIM_STATUS with status MFW_SIM_SUCCESS is received, indicating a correct PUK1 entered
BMI starts is called to check the pin lock. See Ref 6.2.1.3

5.4.3.2 PUK Incorrect



The function *sim_unblock_pin* is called to verify the entered 'puk'.
The event E_SIM_STATUS with status MFW_SIM_PUK_REQ is received indicating an Incorrect PUK1 entered
BMI enters the puk and calls *sim_unblock_pin* again
The event E_SIM_STATUS with status MFW_SIM_SUCCESS is received, indicating a correct PUK1

Else if the number of retries has reached ten MFW_SIM_INVALID_CARD is received and the SIM card is invalidated.

5.4.4 Invalid Card

The full service registration will be stopped.

5.4.5 No Card

The full service registration will be stopped.

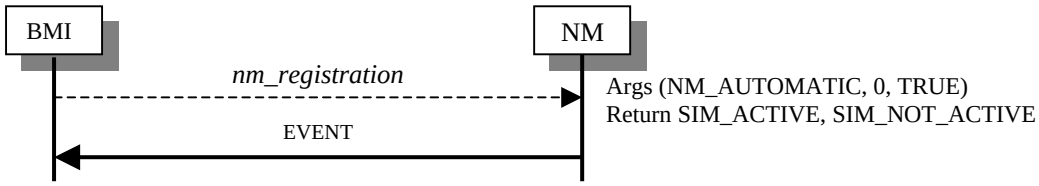
5.5 Network Registration

Once the SIM has been activated the BMI initiates the network registration by selecting either automatic or manual registration. Callback routine used by BMI to receive MFW Events is *network_nm_cb.*

5.5.1 Automatic Registration

The automatic registration can have a choice of either full or limited service.

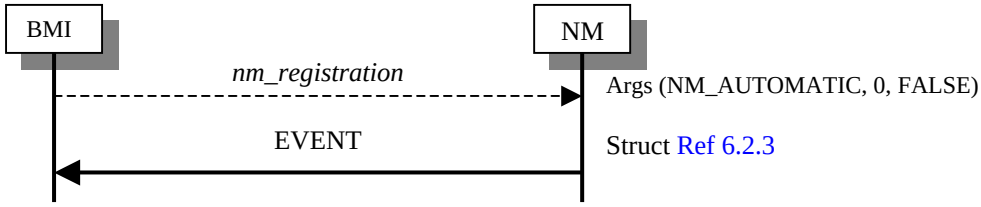
5.5.1.1 Limited Service



The function *nm_registration* is called to start the limited service registration.

If the event E_NM_LIMITED_SERVICE is received the registration has been successful.
Else if the event E_NM_NO_SERVICE is received there is no network so the registration has failed.

5.5.1.2 Full Service



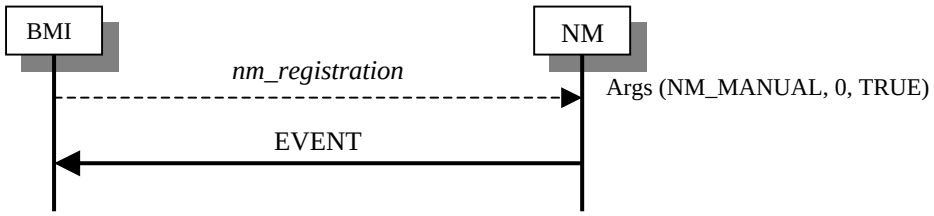
The function *nm_registration* is called to start the full service registration.

If the event E_NM_FULL_SERVICE is received the registration is successful
Else if the event E_NM_NO_SERVICE is received there is no network so registration has failed.
The Structure with the event contains the operator and service names

Note: If SIM authentication has not been done The function returns SIM_NOT_ACTIVE and BMI must start the SIM activation.

5.5.2 Manual Registration

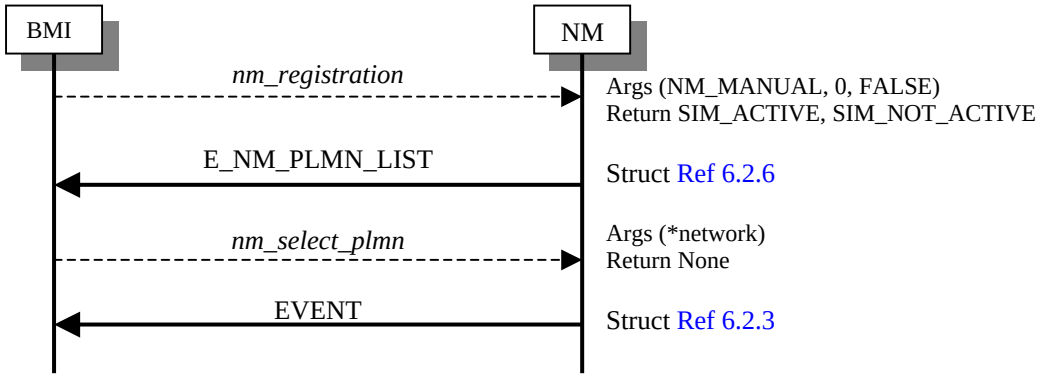
5.5.2.1 Limited Service



The function *nm_registration* is called to start the limited service registration.

If the event E_NM_LIMITED_SERVICE is received the registration has been successful.
Else if the event E_NM_NO_SERVICE is received there is no network so the registration has failed.

5.5.2.2 Full Service



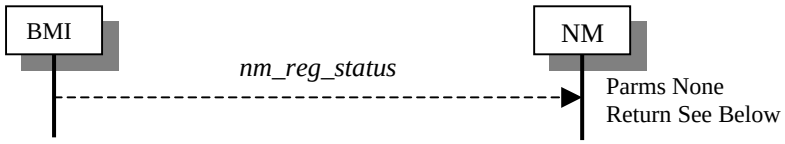
The function *nm_registration* is called to start the full service registration.

If the return is SIM_NOT_ACTIVE then sim activation is required and no event will be sent.
Else if success the event E_NM_PLMN_LIST is received with parameter Structure T_MFW_PLMN_LIST
The mmi has to select one network from the list and call The function *nm_select_plmn*
If the event E_NM_FULL_SERVICE is received the registration is successful

Else if the event E_NM_NO_SERVICE is received there is no network so registration has failed
Else if the event E_NM_LIMITED_SERVICE is received full service has failed so only limited service

The Structure with the event contains the operator and service names.

5.5.3 Registration status



The function *nm_reg_status* is called to examine the current registration status.

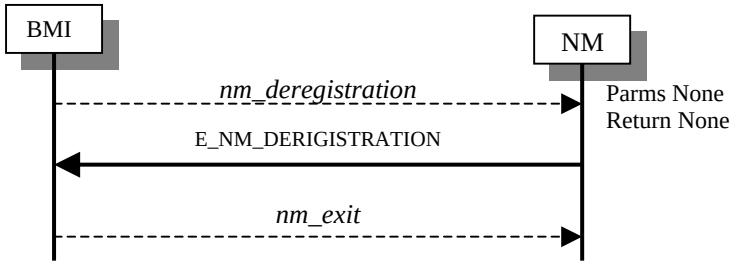
The function returns one of the following:
MFW_FULL_SERVICE
MFW_LIMITED_SERVICE
MFW_NO_SERVICE.

5.5.4 Change of Status

As the mobile is moving, the registration status can change. BMI will be informed about this change by being sent one of the asynchronous events as follows:

E_NM_FULL_SERVICE	Change of full service registration between home PLMN and roaming
EN_LIMITED_SERVICE	Loss of network when in auto registration
EN_NM_LIMITED_SERVICE	Loss of network when in manual registration
E_NM_FULL_SERVICE	When the network is found again

5.5.5 Deregistration

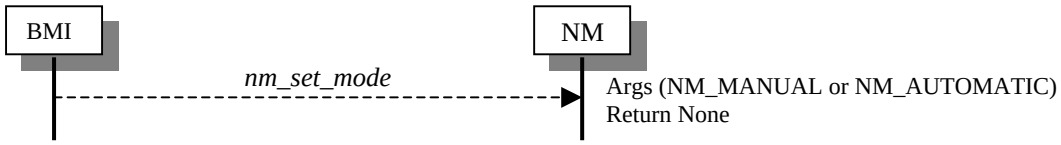


The function *nm_deregistration* is called to deregister the handset.

The event E_NM_DEREGISTRATION is received.
The function *nm_exit* is called to clean up and exit the network

5.5.6 Set Registration Mode

It is possible to change the registration mode between manual and automatic.



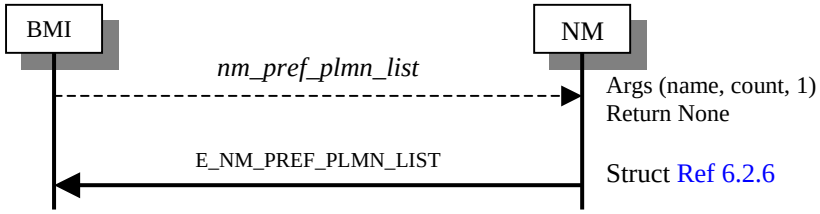
The function *nm_set_mode* is called passing the ‘mode which can have the following values
NM_MANUAL
NM_AUTOMATIC
and the mode is changed

5.6 PLMN lists

5.6.1 Preferred PLMN List

The preferred PLMN list is stored in the SIM card and can be read, edited or read the status of the list.

5.6.1.1 Read Preferred PLMN List

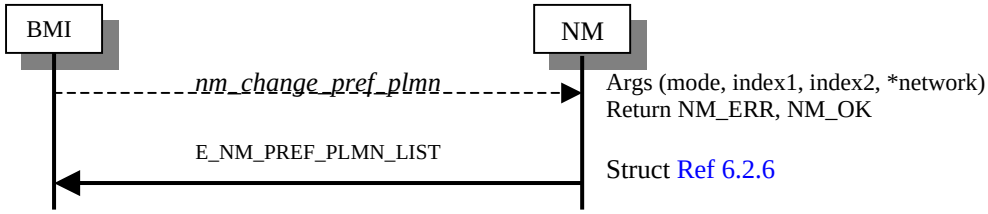


The function *nm_pref_plm_list* is called to request the stored PLMN list

The event E_NM_PREF_PLMN_LIST is received with parameter Structure T_PREF_PLMN_LIST.

5.6.1.2 Edit Preferred PLMN List

The BMI can edit the preferred PLMN list and store this change in the SIM.



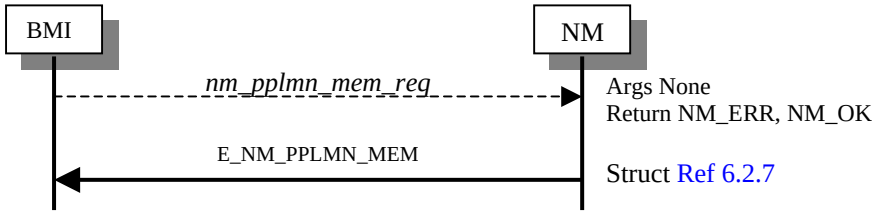
The function *nm_change_pref_plmn* to perform one of the following tasks defined in “mode”:
NEW_PREF_PLMN add a new entry to the end of the list
CHANGE_PREF_PLMN change the place of two entries
INSERT_PREF_PLMN add a new entry to the list at a given position
REPLACE_PREF_PLMN overwrite an existing entry at a given position
DELETE_PREF_PLMN delete an entry from the list

The Args "index" indicates the position of the stored PLMN
The Args "index2" is only used to change the positions of two PLMN's

The event E_NM_PREF_PLMN_LIST is received with parameter Structure T_PREF_PLMN_LIST.

5.6.1.3 Request Preferred PLMN List Memory

BMI can request the available and used PLMN records in the SIM card

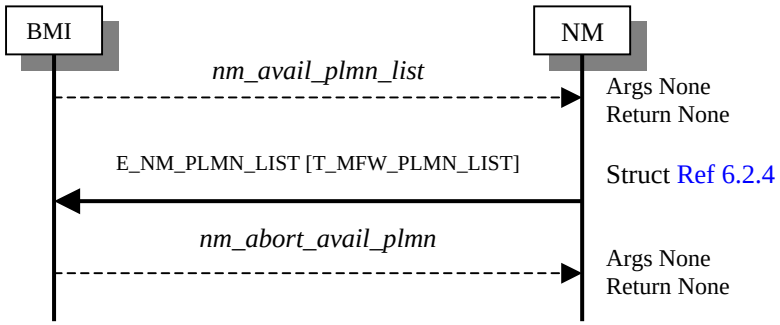


The function *nm_pplmn_mem_req* is called to request the stored PLMN list

The event E_NM_PPLMN_MEM is received with parameter Structure T_PREF_PLMN_LIST

5.6.2 Available PLMN List

The BMI can obtain a list of available networks.



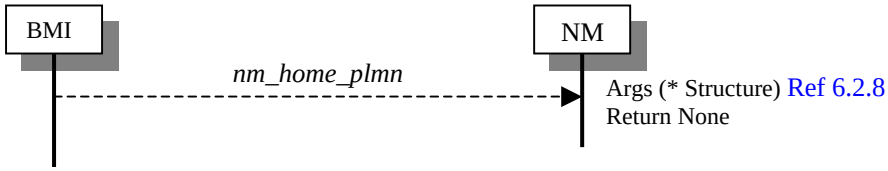
The function *nm_avail_plmn_list* is called to get the available plmn list.

The event E_NM_PLMN_LIST is received with parameter T_MFW_PPLMN_MEM.

Note: At any point the BMI can abort the by calling The function *nm_abort_avail_plmn*

5.6.3 Home PLMN name

This enables the BMI to obtain the home plmn name.



The function *nm_set_mode* is called and the result is written to the passed Structure.

5.7 Security

5.7.1 PIN Verification

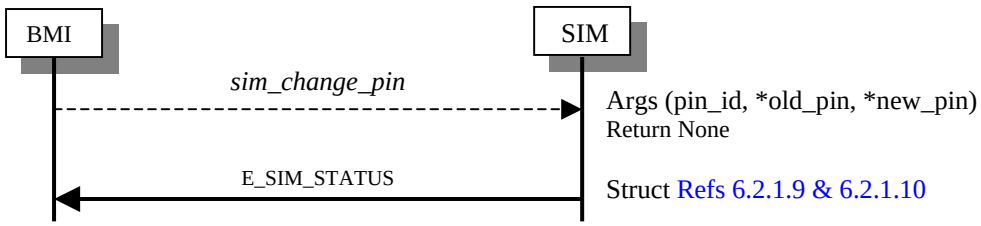
Some operations will be performed under control of PIN1/PIN2. The method of verification of these pins is covered in Ref 5.4.1 to 5.4.2

5.7.2 PIN Unblocking

After the user enters a wrong PIN three times, this PIN is blocked. The user can enter PUK to unblock PIN. The method of unblocking the pin is covered in Ref 5.4.3

5.7.3 PIN Change

From idle only, BMI creates the editing screen to enter the old and new pins.

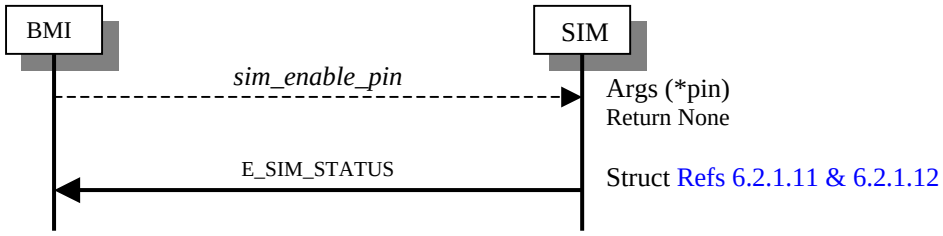


The function *sim_change_pin* is called to request the pin change either for PIN1 or PIN2 by setting pin_id" to either MFW_SIM_PIN1 or MFW_SIM_PIN2.

The event E_SIM_STATUS with parameter MFW_SIM_SUCCESS is received to indicate the pin has changed
Else the event with parameter MFW_SIM_FAILURE is received as the pin could not be changed.

5.7.4 PIN Enable

The user creates the screen menu to enable the PIN.

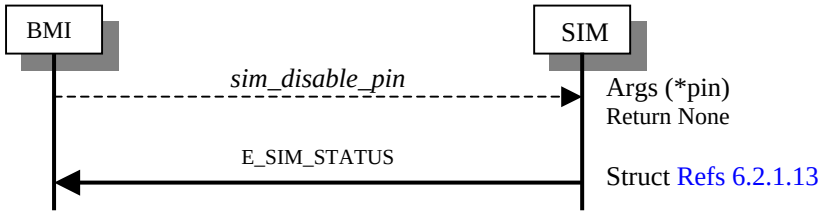


The function *sim_change_pin* is called to enable the PIN

The event E_SIM_STATUS with parameter MFW_SIM_SUCCESS is received to indicate the pin is enabled
Else the parameter is MFW_SIM_FAILURE as the pin could not be enabled.

5.7.5 PIN Disable

The user creates the screen menu to disable the PIN.



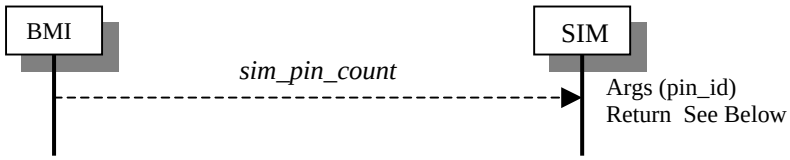
The function *sim_change_pin* is called to disable the PIN

The event E_SIM_STATUS with parameter MFW_SIM_SUCCESS is received to indicate the pin is disabled
Else the parameter is MFW_SIM_FAILURE as the pin could not be disabled.

5.7.6 PIN / PUK Status

5.7.6.1 Count Status

BMI can request the count status of PIN1, PIN2, PUK1, PUK2 at any time.

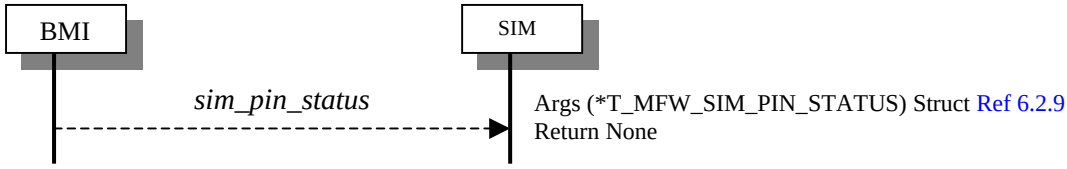


The function *sim_pin_count* is called passing the "pin_id" to request the count status.
Where "pin_id" has the following possible values
MFW_SIM_PIN1
MFW_SIM_PIN2
MFW_SIM_PUK1
MFW_SIM_PUK2

The function returns the count if successful else the failed value –1.

5.7.6.2 State Status

BMI can request the pin state at any time.



The function *sim_pin_status* is called setting "type" in the passed Structure and the result is written into the Structure with "set" as the PIN state and "stat" as whether a pin is required or not.
"set" has the possible values
MFW_SIM_ENABLE
MFW_SIM_DISABLE

"stat" has the possible values
MFW_SIM_NO_PIN
MFW_SIM_PIN_REQ
MFW_SIM_PIN2_REQ

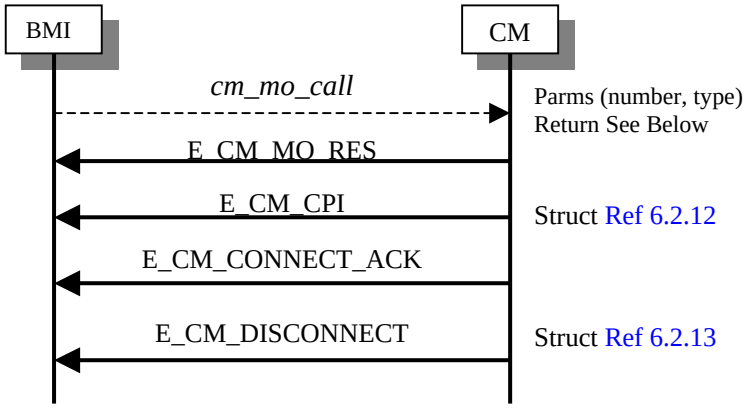
5.8 Call

To attempt an outgoing call the parameter “number” can be any string such as a keystroke sequence. MFW informs BMI about the type of the given string. Callback routine used by BMI to receive MFW Events is *callCmEvent.*
When the user enters a call number string the MFW performs checks (such as whether one call is in set up state) before starting the call. The call progress is as follows.

5.8.1 Outgoing Call

5.8.1.1 Call setup successful

BMI attempts to set up a call that is successful.



The function *cm_mo_call* is called and returns one of the following codes indicating success.

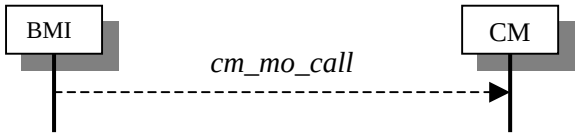
CM_OK	The call setup is started
CM_EC	The emergency call setup is started

The event E_CM_MO_RES is received once the call has started (this event includes the call identifier)
Followed by the event E_CM_CPI indicating call progress
Finally the event E_CM_CONNECT_ACK is received when the connection is made (event includes call identifier)
Else if the connection has failed the event E_CM_DISCONNECT event is received.

Note: If the connection has failed and the redial is on, MFW starts redial and after the maximum redial attempts is reached the number is stored in the blacklist.

5.8.1.2 Call setup fails

BMI attempts to set up a call that fails.



The function *cm_mo_call* is called and returns one of the following codes indicating failure.

CM_ERROR	number cannot be completed as dialed.
CM_BLACKLIST	number is blacklisted.
CM_BUSY	no connection because call set up is being done
CM_ACM_MAX	ACM reaches maximal value
CM_CFDN_ERR	check FDN number error
CM_CBDN_ERR	check BDN number error
CM_2CALLS_ALREADY_ERR	two calls already, no third outgoing call allowed

5.8.2 Redial

The function *cm_set_redial* has to be called during the initialization process to set the auto redial mode to either auto or manual. The BMI has to start the auto redial only once whereas the user has to start each manual redial.

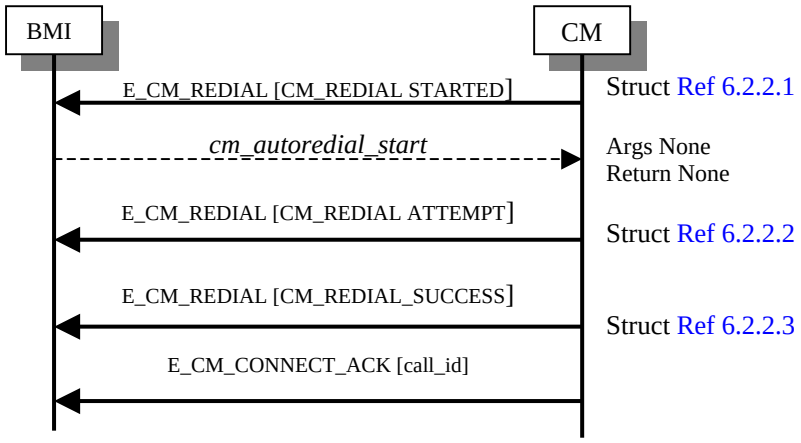
If the redial fails the BMI receives the event E_CM_REDIAL REDIAL with the appropriate cause in the parameter "redial_mode".

CM_REDIAL_NO_SERVICE	mobile out the cell
CM_REDIAL_INCOMING	redial abort because of incoming call
CM_REDIAL_ABORTED	abort redial from BMI
CM_REDIAL_OUTGOING	redial abort because of outgoing call
CM_REDIAL_STOP	stop redialing (outside of category)

5.8.2.1 Autoredial

5.8.2.2 Auto redial successful

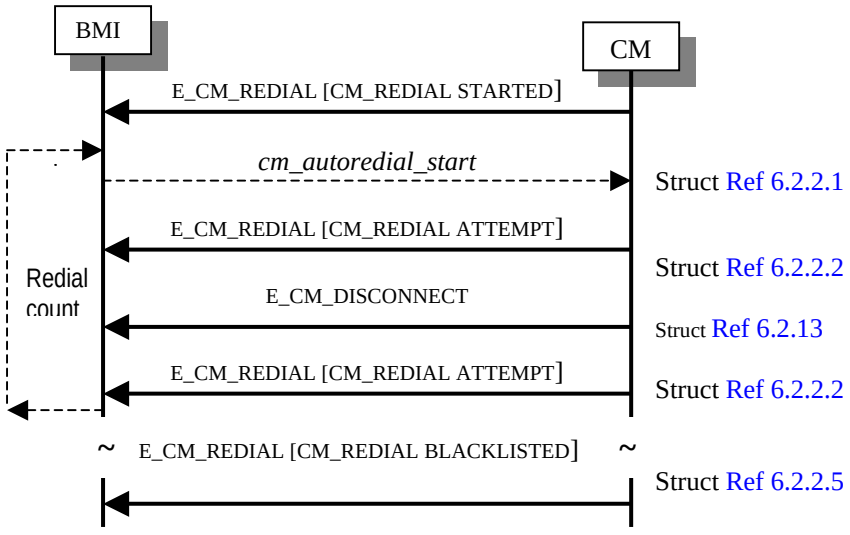
Following a disconnection event E_CM_DISCONNECT auto redial is started and is successful.



The event E_CM_READIAL [CM_REDIAL_STARTED] is received requesting auto redial.
The BMI calls *cm_autoredial_start* to start the redial.
The event E_CM_REDIAL [CM_REDIAL_ATTEMPT] is received indicating redial is proceeding
Followed by the event E_CM_REDIAL [CM_REDIAL_SUCCESS] indicating redial has been successful.
Then the event E_CM_CONNECT_ACT is received when the connection is made (event includes call identifier).

5.8.2.3 Auto redial fails

Following a disconnection event E_CM_DISCONNECT auto redial is started and fails.



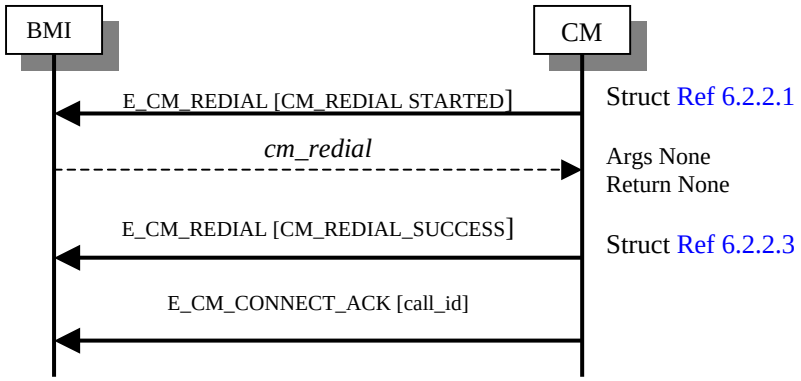
The event E_CM_READIAL [CM_REDIAL_STARTED] is received requesting auto redial.
The BMI calls *cm_autoredial_start* to start the redial.
The event E_CM_REDIAL [CM_REDIAL_ATTEMPT] is received indicating redial is proceeding
Followed by the event E_CM_DISCONNECT indicating that the redial has failed

The redial is automatically started again.
The event E_CM_REDIAL [CM_REDIAL_SUCCESS] is received indicating redial has been successful.
Followed by the event E_CM_CONNECT_ACT when the connection is made.
Else if the redial count has been reached
The event E_CM_REDIAL [CM_REDIAL_BLACKLISTED] is received and the number is written into the blacklist.

5.8.2.4 Manual Redial

5.8.2.4.1 Manual redial successful

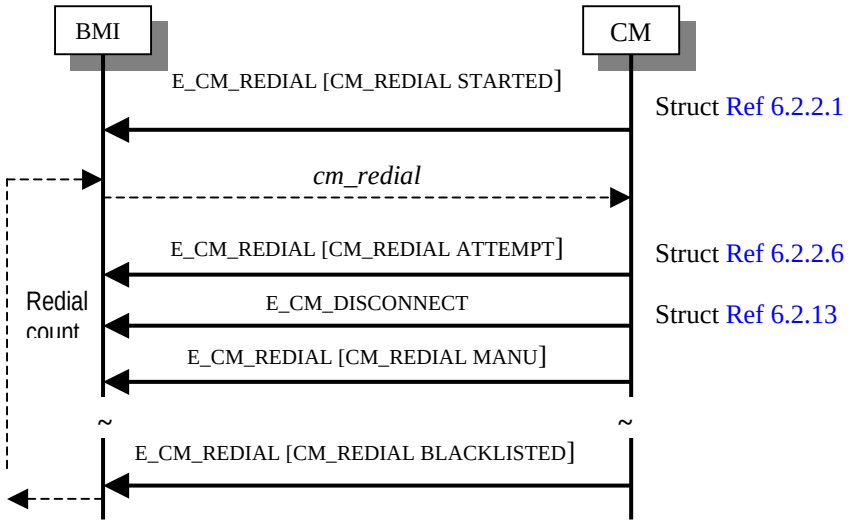
Following a disconnection event E_CM_DISCONNECT manual redial is started and is successful.



The event E_CM_READIAL [CM_REDIAL_STARTED] is received requesting manual redial.
The BMI calls *cm_redial* to start the redial.
The event E_CM_REDIAL [CM_REDIAL_MANU] is received indicating redial is proceeding
Followed by the event E_CM_REDIAL [CM_REDIAL_SUCCESS] indicating redial has been successful.
Then the event E_CM_CONNECT_ACT is received when the connection is made (event includes call identifier).

5.8.2.4.2 Manual redial fails

Following a disconnection event E_CM_DISCONNECT auto redial is started and fails.

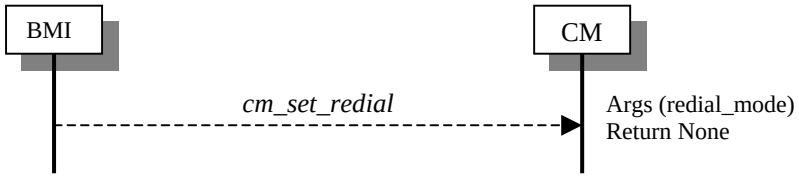


The event E_CM_READIAL [CM_REDIAL_STARTED] is received requesting manual redial.
The BMI calls *cm_redial* to start the redial.
The event E_CM_REDIAL [CM_REDIAL_MANUAL] is received indicating redial is proceeding
The event E_CM_DISCONNECT is received indicating that the redial has failed
Followed by the event E_CM_READIAL [CM_REDIAL_STARTED] requesting the next manual redial.
The BMI calls *cm_redial_start* to start the redial.
The event E_CM_REDIAL [CM_REDIAL_SUCCESS] is received indicating redial has been successful.
Followed by the event E_CM_CONNECT_ACT when the connection is made.

Else if the redial count has been reached
The event E_CM_REDIAL [CM_REDIAL_BLACKLISTED] is received and the number is written into the blacklist.

5.8.2.5 Redial Mode

BMI can set and request the redial mode



The function *cm_set_redial* is called passing one of the following mode values:
CM_REDIAL_OFF
CM_REDIAL_AUTO
CM_REDIAL_MANU

To request the redial mode The function *cm_get_redial_mode* is called and returns one of the above values.

5.8.2.6 Bearer Parameter for Mobile Originated Calls

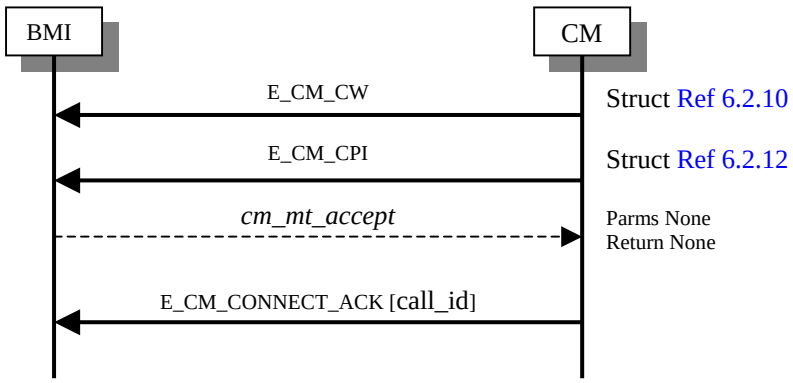
BMI can use The function *cm_set_bc_para* to set the bearer parameters for the data call and fax call. In the current application, BMI can only set the baud rate, the bearer service and the connection element.

5.8.3 Incoming Call

When an incoming call arrives BMI can accept or reject the call.

5.8.3.1 Call Accept

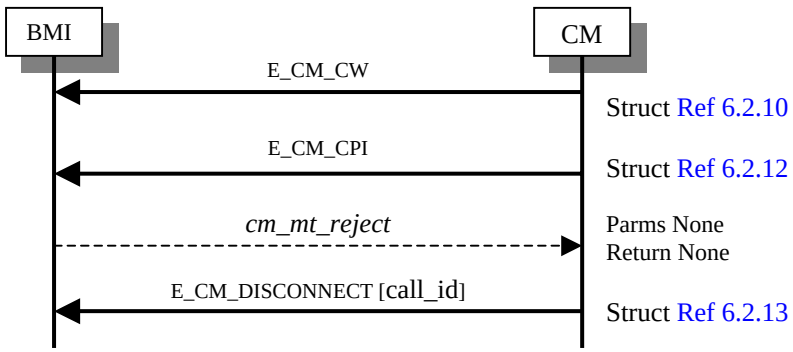
BMI accepts the incoming call.



The event E_CM_CW is received indicating an incoming call
This is followed by the event E_CM_CPI indicating the call is proceeding and giving the audio status
The function *cm_mt_accept* is called to accept the call
When the connection is successful the event E_CM_CONNECT_ACK is received (event includes call identifier)
Else if the connection fails the event E_CM_DISCONNECT is received with a Structure indicating the cause.

5.8.3.2 Call Reject

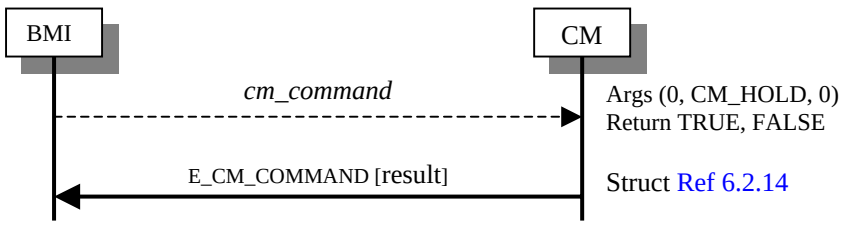
BMI rejects the call.



The event E_CM_CW is received indicating an incoming call
This is followed by the event E_CM_CPI indicating the call is proceeding and giving the audio status
The function *cm_mt_reject* is called to reject the call
When the call is disconnected the event E_CM_DISCONNECT is received.

5.8.4 Hold the Call

The BMI can change the call status by putting the call on hold.

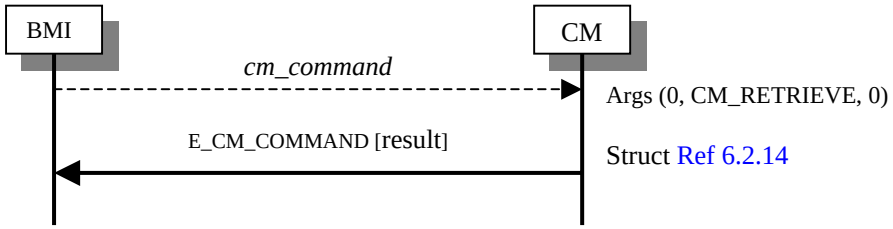


The function *cm_command* is called is called to check whether the call can be put on hold.

If the return is FALSE the attempt ends else the event E_CM_COMMAND is received with command parameter CM_HOLD and result parameter Present or NotPresent indicating success or fail.

5.8.4.1 Call Retrieve

The BMI can retrieve a held call.

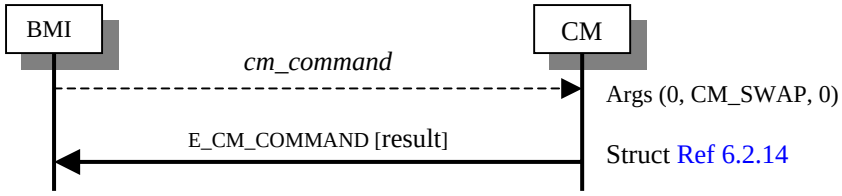


The function *cm_command* is called is called to check whether the call can be retrieved.

If the return is FALSE the attempt ends else the event E_CM_COMMAND is received with command parameter CM_RETREIVE and result parameter Present or NotPresent indicating success or fail.

5.8.4.2 Call Swap

The BMI can swap between an active and a held call.

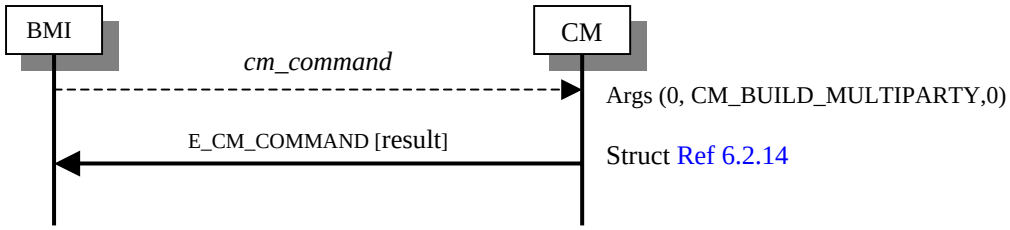


The function *cm_command* is called is called to check whether the calls can be swapped.

If the return is FALSE the attempt ends else the event E_CM_COMMAND is received with command parameter CM_SWAP and result parameter Present or NotPresent indicating success or fail.

5.8.4.3 Build Multiparty Call

The BMI can set up a multi party call by adding the held call.

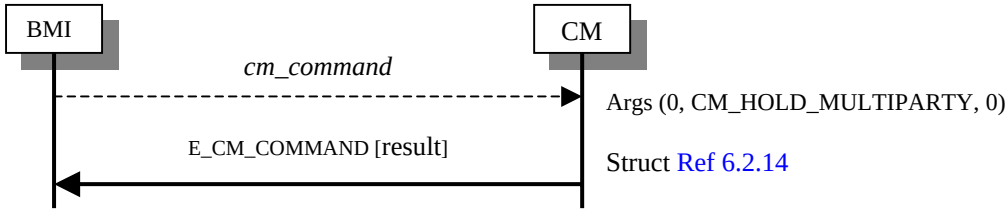


The function *cm_command* is called is called to check whether the call can be added to the multiparty.

If the return is FALSE the attempt ends else the event E_CM_COMMAND is received with command parameter CM_BUILD_MULTIPARTY and result parameter Present or NotPresent indicating success or fail.

5.8.4.4 Hold Multiparty Call

The BMI can put the active multi party call on hold.

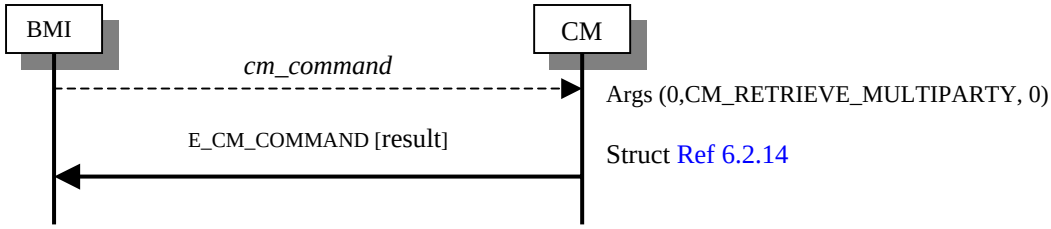


The function *cm_command* is called is called to check whether the multi party call can be held.

If the return is FALSE the attempt ends else the event E_CM_COMMAND is received with command parameter CM_HOLD_MULTIPARTY and result parameter Present or NotPresent indicating success or fail.

5.8.4.5 Retrieve Multiparty Call

The BMI can retrieve the held multi party call.

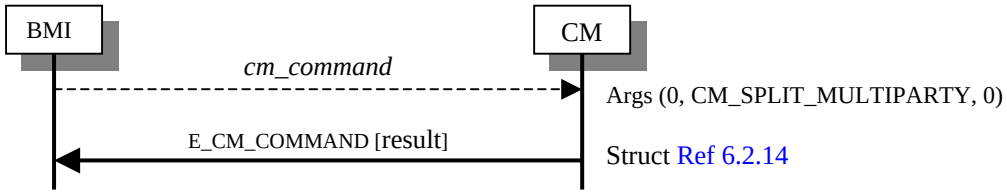


The function cm_command is called to check whether the held multi party call can be made active.

If the return is FALSE the attempt ends else the event E_CM_COMMAND is received with command parameter CM_RETRIEVE_MULTIPARTY and result parameter Present or NotPresent indicating success or fail.

5.8.4.6 Split Multiparty Call

The BMI can split the multi party call.

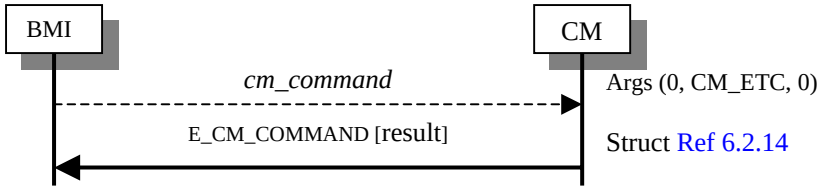


The function *cm_command* is called to check whether the multi party call can be split.

If the return is FALSE the attempt ends else the event E_CM_COMMAND is received with command parameter CM_SPLIT_MULTIPARTY and result parameter Present or NotPresent indicating success or fail.

5.8.4.7 Explicit Call Transfer

The BMI can connect two calls and disconnect the subscriber from both.

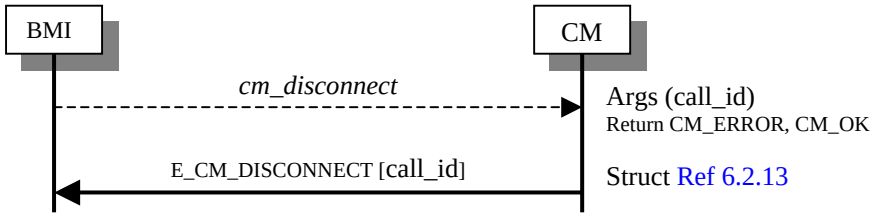


The function *cm_command* is called to check whether the explicit call can be made.

If the return is FALSE the attempt ends else the event E_CM_COMMAND is received with command parameter CM_ETC and result parameter Present or NotPresent indicating success or fail.

5.8.5 Ending the Call

The call can be terminated by the BMI or by the network.



The function *cm_disconnect* is called to terminate the call passing the call identification
If the call id is zero then the BMI wants to disconnect the multi party call

The event E_CM_DISCONNECT is received indicating that the call has ended

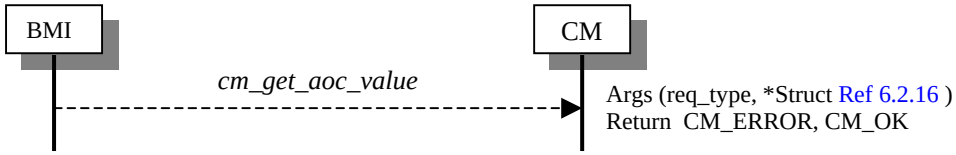
Note: Event E_CM_DISCONNECT can be received asynchronously indicating that network has ended the call.

5.9 Advice of Charge

The BMI has the ability to set and request, the various parameters associated with the advice of charge.

5.9.1 Request AoC

BMI requests a parameter associated with the advice of charge.



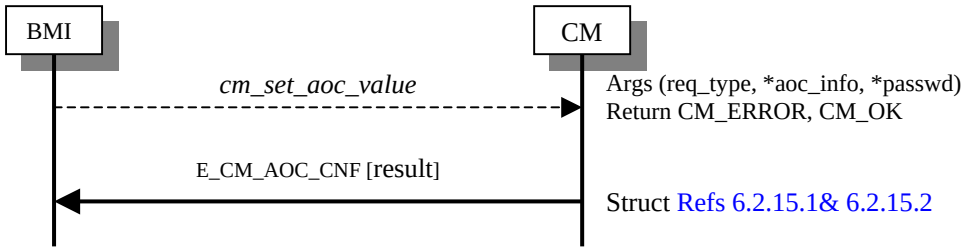
The function *cm_get_aoc_value* is called to request a parameter defined in “reg_type” and the result is written into the passed Structure.

The following AoC values can be requested:

CM_AOC_PUCT	Price pro unit and currency table
CM_AOC_ACMMAX	maximum value of accumulated Call Meter
CM_AOC_ACM	Accumulated Call Meter
CM_AOC_CCM	Current Call Meter
CM_AOC_TIMERS	The timer of current call
CM_AOC_CCM_PUCT	Current Call Meter using PUCT
CM_AOC_ACM_PUCT	Accumulated Call Meter using PUCT
CM_AOC_RAW_PUCT	Raw PUCT

5.9.2 Setting AoC

BMI sets a parameter associated with the advice of charge.



The function *cm_set_aoc_value* is called to set a parameter defined in “reg_type”.

The following AoC values can be set:

CM_AOC_PUCT	Price pro unit and currency table
CM_AOC_ACMMAX	maximum value of accumulated Call Meter
CM_AOC_ACM	reset ACM

The event E_CM_AOC_CNF with parameter CM_AOC_CONF_OK is received to indicate the value is set
Else the parameter indicates the reason for failure.

5.10 Short Message Service

5.10.1 Initialisation

At power up the event E_SMS_READY is received indicating that the event handler is ready.
Callback routine used by BMI to receive MFW Events is *smsidle_sms_cb*.

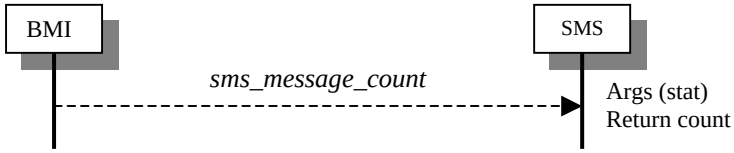
Note: At any stage the asynchronous message E_SMS_MEM_FULL can be received indicating that the SMS memory is full. Messages must be deleted to use SMS services.

5.10.2 Status

The following status information is available on request.

5.10.2.1 Message Count

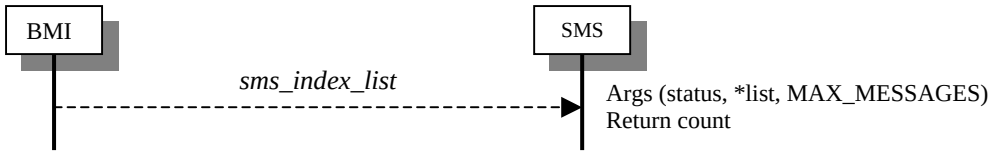
The BMI can request the various SMS message counts



BMI calls function s*ms_message_count* which returns the count associated with one of the passed parameters:
MFW_SMS_UNREAD
MFW_SMS_READ
MFW_SMS_STOR_UNSENT
MFW_SMS_STOR_SENT,
MFW_SMS_MO,
MFW_SMS_MT
MFW_SMS_ALL
MFW_SMS_NotVOICE,
MFW_SMS_VOICE

5.10.2.2 Index List

The BMI can request the SMS index list

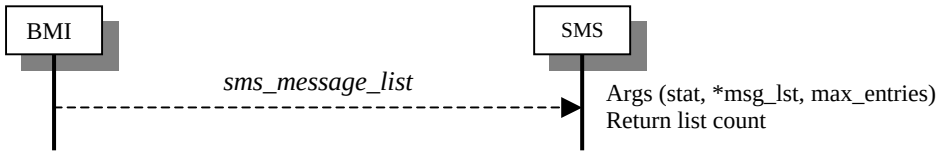


BMI calls function s*ms_index_list* which returns the list associated with one of the passed parameters:
MFW_SMS_UNREAD
MFW_SMS_READ
MFW_SMS_STOR_UNSENT
MFW_SMS_STOR_SENT,
MFW_SMS_MO,
MFW_SMS_MT
MFW_SMS_ALL
MFW_SMS_NotVOICE,
MFW_SMS_VOICE

See Ref 6.2.17 for parameter *list*

5.10.2.3 Message List

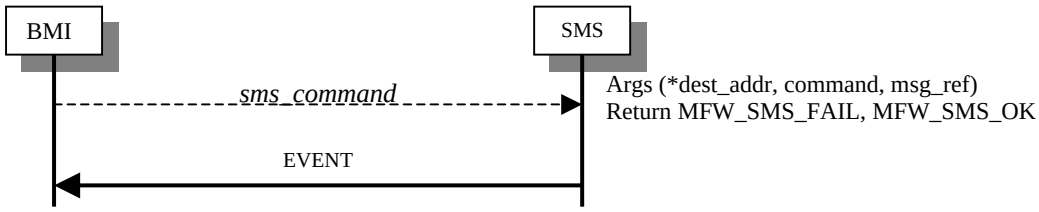
The BMI can request the Address, index, message status, time etc of the SMS message list.



The function s*ms_message_list* is called which stores information in the passed Structure “msg_lst” (Ref 6.2.18) for one of the passed parameters above.

5.10.2.4 Status of Message

The BMI can request a status report or an enquiry for a message



The function s*ms_command* is called passing one of the following “command” parameters:

SMS_ENQ_PREV_SMS	Starts an enquiry to previously sent short message
SMS_CAN_STAT_REQ	Cancels status report request
SMS_DEL_PREV_SMS	Deletion of previous sent short message
SMS_REQ_STAT_REP	Requests status report for SMS message

If the event E_SMS_CMD_AVAIL is received the command has been successful.

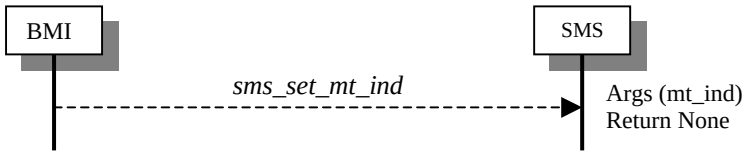
Else if the event E_SMS_ERR is received the command has failed.

5.10.3 Configuration

BMI can configure the following SMS parameters.

5.10.3.1 Set New Message Indication

The BMI can select the method of how it will be informed of the arrival of an SMS message both for mobile terminated and cell broadcast



The function s*ms_set_mt_ind* is called to select how it will be informed of the reception of SMS messages.

For Mobile Terminated messages the parameter passed can have one of the following values.

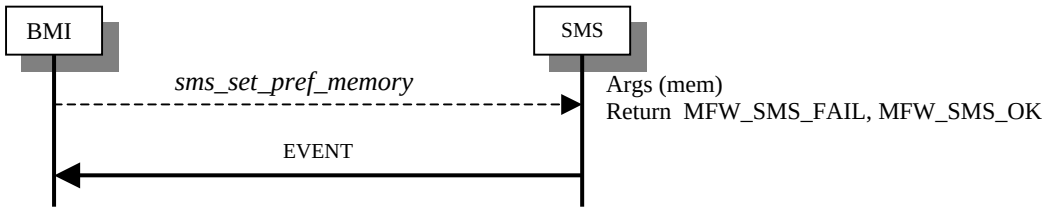
MT_IND_IDX	the message is stored in memory and the index is given
MT_IND_NO_IDX	the message is stored in memory but no index is given
MT_IND_MSG	the message is not stored in memory

On reception of a message the event E_SMS_MT_RECEIVED [T_MFW_SMS_ID] is received with the index if appropriate.

For Cell Broadcast messages the parameter passed can have one of the following values.
MT_CB_IDX the message is stored in memory and the index is given
MT_CB_NO_IDX the message is stored in memory but no index is given
MT_CB_MSG the message is not stored in memory
On reception of a message the event E_SMS_CB_RECEIVED is received with the index if appropriate.
Note: In the current application the CB message cannot be saved in memory.

5.10.3.2 Set Preferred Message Storage

BMI sets the message storage to either memory or SIM

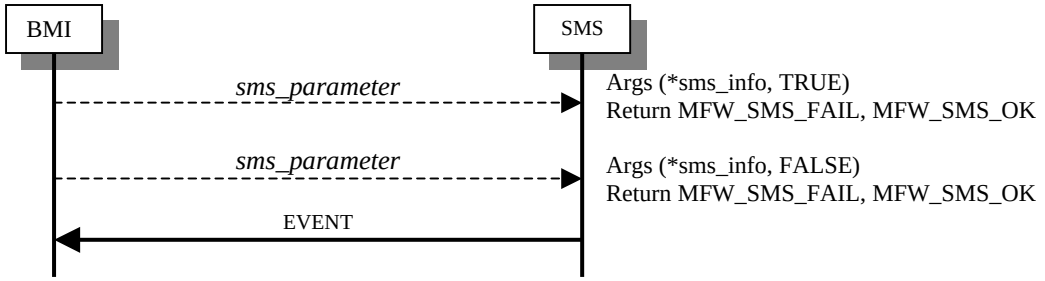


The function s*ms_set_pref_memory* is called passing one of the following "mem" parameters:
MFW_MEM_ME preferred message storage is set to memory
MFW_MEM_SIM message storage is set to SIM.

If the event E_SMS_MEM is received the command has been successful
Else if the event E_SMS_ERR is received the command has failed.

5.10.3.3 Set SMS Parameter

BMI can change the settings of the SMS parameters

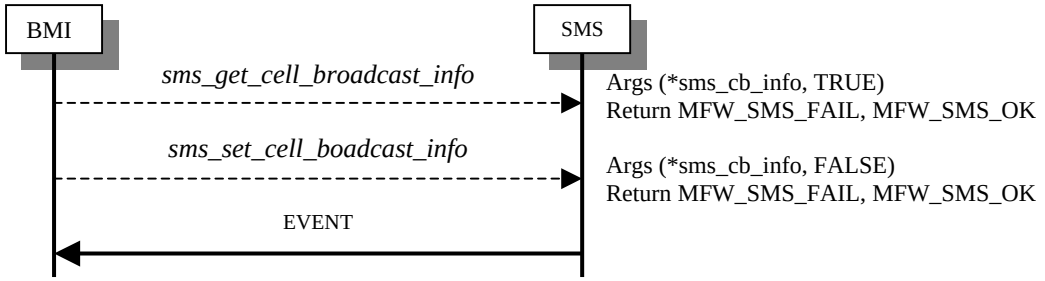


The function s*ms_parameter* is called to read the parameters into the passed Structure Ref 6.2.20
The function *sms_parameter* is then called to set one of the parameters

If the event E_SMS_OK is received the command has been successful
Else if the event E_SMS_ERR is received the command has failed.

5.10.3.4 Set Cell Broadcast Parameter

BMI can change the settings of the cell broadcast parameters



The function s*ms_get_cell_broadcast_info* is called to read the parameters into the passed Structure Ref 6.2.21
BMI then calls The function *sms_cell_set_broadcast_info* to set one of the parameters

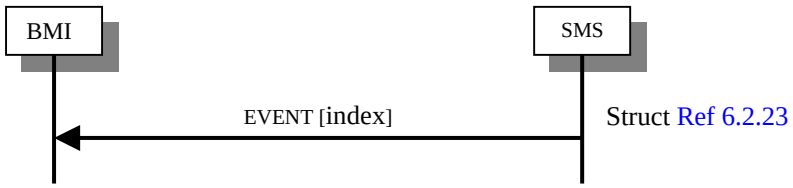
If the event E_SMS_OK is received the command has been successful
Else if the event E_SMS_ERR is received the command has failed.

5.10.4 Incoming Message

The message sent upon the arrival of an SMS message depends upon the method of indication setup above.

5.10.4.1 Message Indication Received

Only the indication is received.



For a mobile terminated message the event E_SMS_MT_RECEIVED is received
else for a cell broadcast message the event E_SMS_CB_RECEIVED is received
in either case the event parameter gives the “index” of the message in memory.

5.10.4.2 Message Received

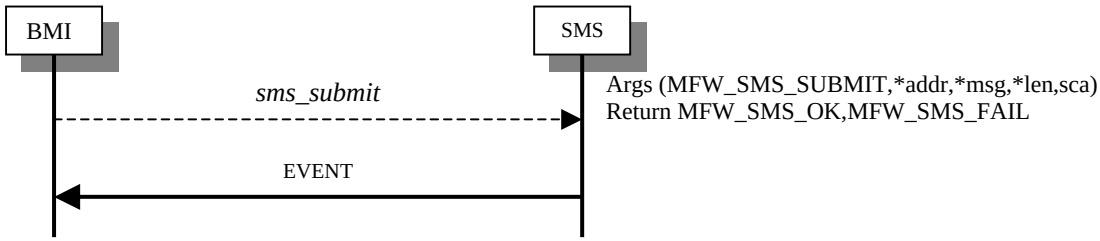
The message is received in its entirety.

For a mobile terminated message the event E_SMS_MT is received
else for a cell broadcast message the event E_SMS_CB is received

5.10.5 Outgoing Message

5.10.5.1 From the user

BMI creates a screen to enter the message text and the destination address.

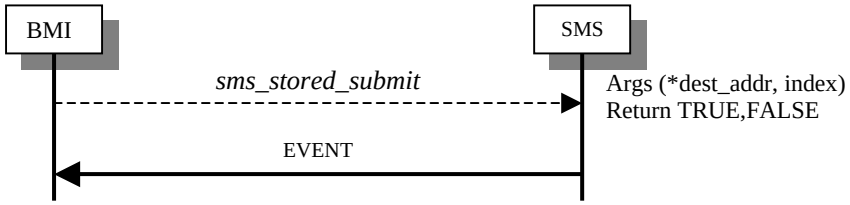


The function s*ms_submit* is called where “type” indicates a new or reply message and “sca” the centre address.
If “sca” is NULL the service centre address, which is stored in the memory is used.

If the event E_SMS_MO_AVAIL is received the send has been successful.
Else if the event E_SMS_ERR is received the send has failed.

5.10.5.2 From Memory

BMI creates a screen to enter the message index and the destination address.

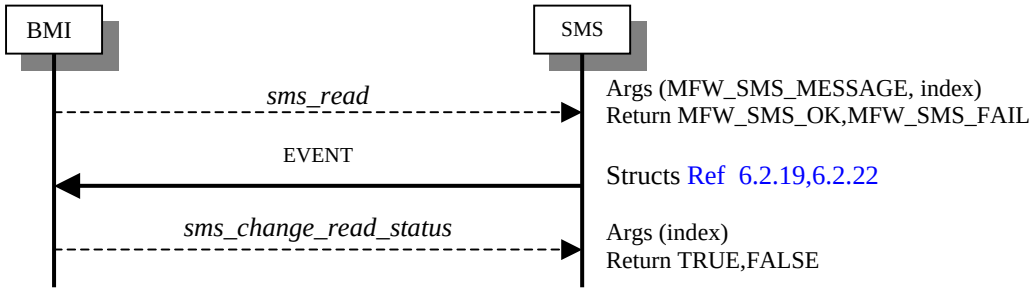


The function s*ms_stored_submit*. is called

If the event E_SMS_MO_AVAIL is received the send has been successful.
Else if the event E_SMS_ERR is received the send has failed.

5.10.6 Read a Message

BMI reads a mobile originated or terminated message.



The function s*ms_read* is called to read the message.

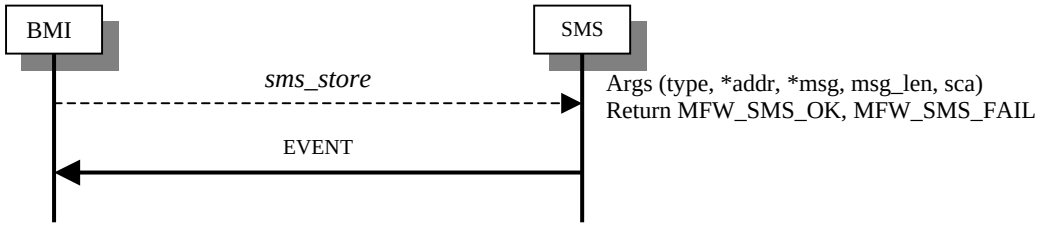
If the event E_SMS_MT is received the read of the mobile terminated message has been successful
else If the event E_SMS_MO is received the read of the mobile originated message has been successful
else if the event E_SM_ERR is received the read has been unsuccessful
On a successful read The function s*ms_change_read_status* is called.

5.10.7 Store a Message

The method of storage is different for a mobile originated or mobile terminated message.

5.10.7.1 Mobile Originated

BMI stores a mobile originated message.

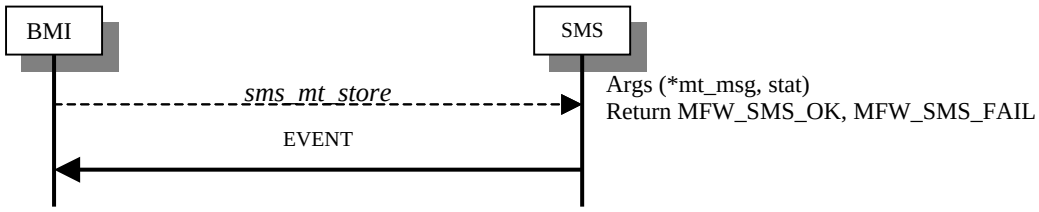


The function s*ms_store* is called

If the event E_SMS_SAVE_AVAIL is received the store has been successful.
Else if the event E_SMS_ERR is received the store has failed.

5.10.7.2 Mobile Terminated

BMI stores a mobile originated message

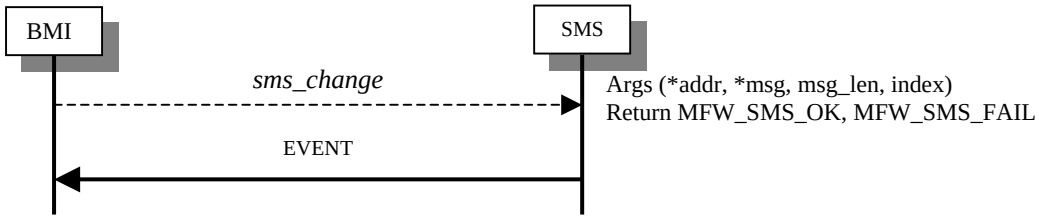


The function s*ms_mt_store* is called

If the event E_SMS_SAVE_AVAIL is received the store has been successful.
Else if the event E_SMS_ERR is received the store has failed.

5.10.8 Change a Message

BMI creates the screen to edit the message and reads the message to be changed.

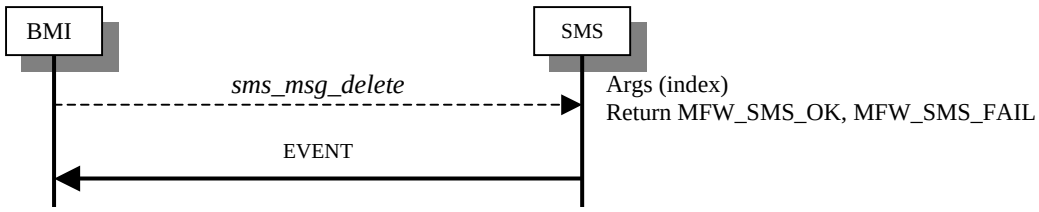


The function s*ms_change* is called

If the event E_SMS_SAVE_AVAIL is received the change has been successful.
Else if the event E_SMS_ERR is received the changed has failed.

5.10.9 Delete a Message

BMI deletes a stored message



The function s*ms_msg_delete* is called

If the event E_SMS_OK is received the store has been successful.
Else if the event E_SMS_ERR is received the store has failed.

5.11 Phonebook

There are ten phonebooks currently supported as shown in the table below.

Phonebook	Meaning	Logical index	Physical index	Letter	Number
ECC	Emergency Call Number	•	•		•
ADN	Abbreviated Dialing Number	•	•	•	•
FDN	Fixed Dialing Number	•	•	•	•
BDN	Barred Dialing Number	•	•	•	•
LRN	Last Received Number		•		
LDN	Last Dialled Number		•		
LMN	Last Missed Number		•		
SDN	Service Dialing Number	•	•	•	•
UPN	User Person Number	•	•	•	•
ADN + FDN	Merged ADN & FDN	•		•	•

Entries in these phonebooks can be read, saved and deleted. In addition entries can be searched using the logical index, physical index, letter or number. The table shows which option is supported for each phonebook.

The results of read, write, find and delete operations are returned to BMI immediately.

On power up BMI receives the event E_PHB_BUSY and once the phonebook has been initialised the event E_PHB_READY is sent. The phonebook is now ready for use.

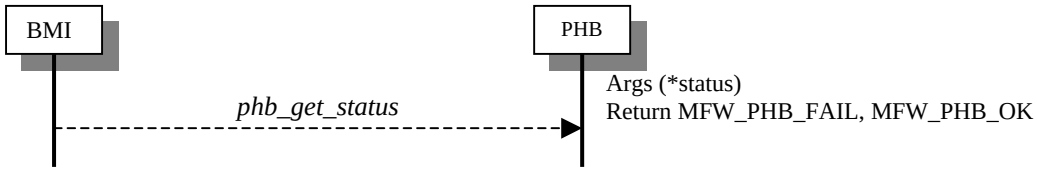
During an update of the phonebook BMI receives the event E_PHB_BUSY and once this completes the event E_PHB_READY is sent. The user can access the phonebook again.

5.11.1 Status

The following status information on the phonebook is available on request.

5.11.1.1 Request Phonebook Memory Information

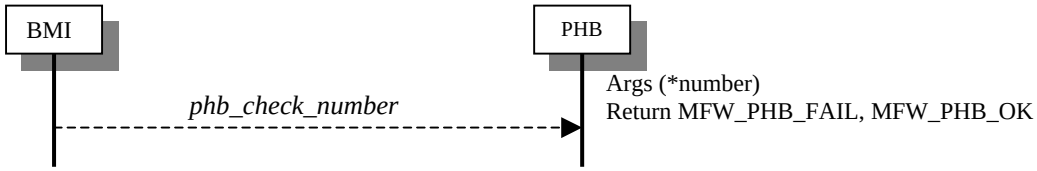
BMI can request the status of the phonebook memory



The function *phb_get_status* is called passing the parameter “status” in which the information about the phonebook is written see Structure Ref 6.2.27

5.11.1.2 Check FDN Number

BMI can check whether a given number is in the FDN phonebook



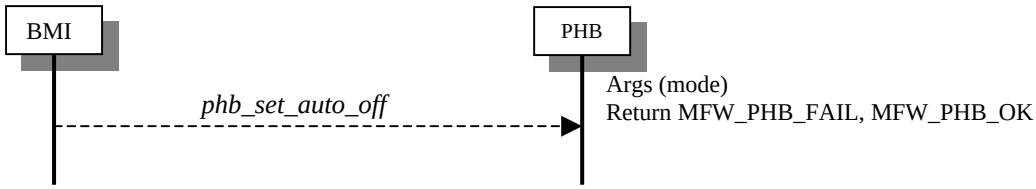
The function *phb_check_number* is called passing the parameter “number”

5.11.2 Configuration

BMI can configure the following Phonebook parameters.

5.11.2.1 Method of Storage for LDN / LRN / LMN

Numbers can be stored in the LDN, LRN and LMN phonebook either automatically or from BMI. The choice is user selectable.

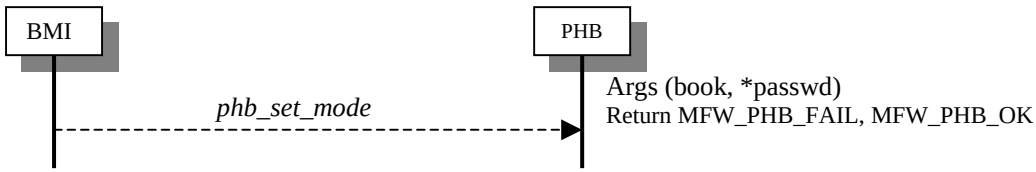


The function *phb_set_auto_off* is called passing the "mode" parameter set to one of the following values:

PHB_AUTO_ON	numbers will be entered into phonebook from the lower layer
PHB_AUTO_OFF	BMI should enter the number into phonebook after the call is set up, accepted or rejected

5.11.2.2 Switch Between Restricted and Unrestricted Mode

BMI can switch between the restricted FDN mode and the unrestricted ADN mode. This has to be done under the control of PIN2.



The function *phb_set_mode* is called passing the following parameters:

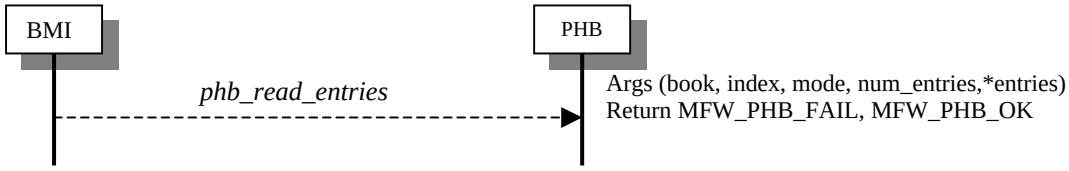
"book"	defined in Read Entries above
"passwd"	the PIN2 password

If PIN2 is incorrect the BMI will not be informed about the PIN2 count

Note: As an alternative, BMI can verify the PIN as in Ref 5.4.2 and then call *phb_set_mode.*

5.11.3 Read Entries

BMI can read any of the phonebook entries



The function *phb_read_entries* is called to read the phonebook entries passing the following parameters:
"book" is one of the ten phonebooks PHB_*
where * has one of the values EEC, ADN, FDN, BDN, SDN, LRN, LDN, LMN, UPN or ADN_FDN
"index" indicates the start of the entries to be read

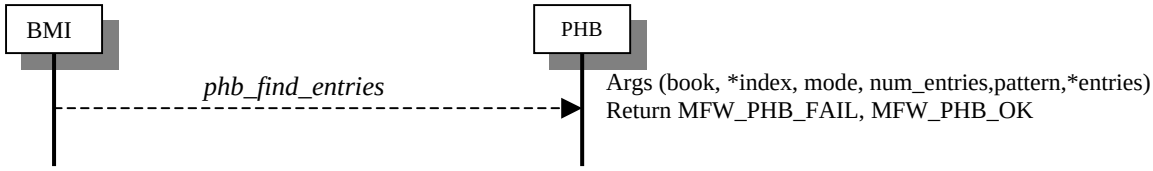
"mode" indicates the read order and has one of the following values:

MFW_PHB_INDEX	ordered according to the logical index
MFW_PHB_NUMBER	ordered according to the number
MFW_PHB_ALPHA	ordered according to the letter
MFW_PHB_PHYSICAL	ordered according to the physical index

"num_entries" is the number of entries to read
"entries" is an array of Structures into which the entries are written see Refs 6.2.25 & 6.2.26

5.11.4 Find Entries

BMI can search for a phonebook entry

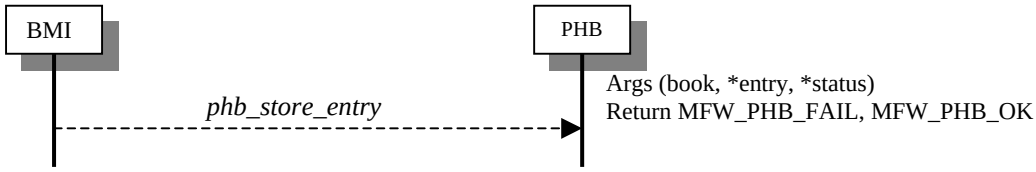


The function *phb_find_entries* is called to find the phonebook entries passing the following parameters:
"book", "mode", "num_entries" and "entries" are the same as Read Entries above
"pattern" is the search string which can be a letter string or a number.
"index" is the index of the first matched entry

5.11.5 Write an Entry

5.11.5.1 Phonebooks excepting FDN

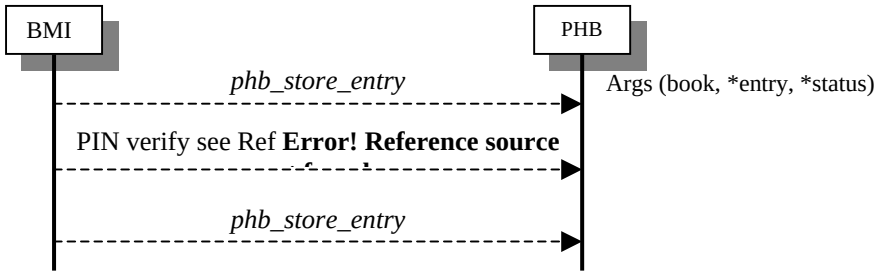
BMI can write an entry to the phonebook



The function *phb_store_entry* is called to store a phonebook entry passing the following parameters:
"entry" is the entry to be written see Structure Ref 6.2.26
"status" information about the phonebook is written to this Structure see Structure Ref 6.2.27

5.11.5.2 FDN Phonebook

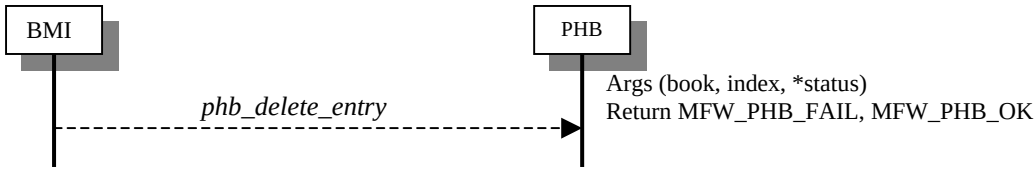
An FDN entry can be stored only under the control of PIN2



The function *phb_store_entry* is called which returns MFW_PHB_FAIL as PIN2 is needed
BMI creates the edit screen to enter PIN2
The PIN2 is verified and *phb_store_entry* called again to store the FDN entry.

5.11.6 Delete an Entry

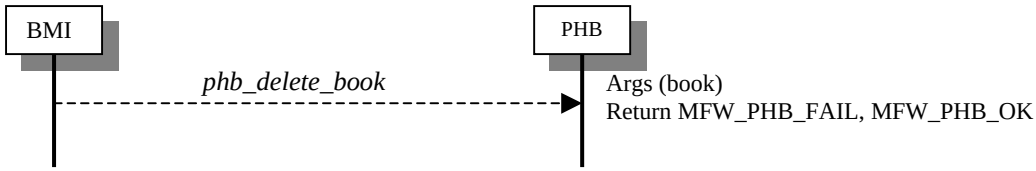
BMI can delete an entry from the phonebook



The function *phb_delete_entry* is called to store a phonebook entry passing the following parameters:
"book" the phonebook of the entry being deleted
"index" is the physical index of the entry to be deleted
"status" information about the phonebook is written to this Structure see Structure Ref 6.2.27

5.11.7 Delete a Phonebook

BMI can delete a phonebook



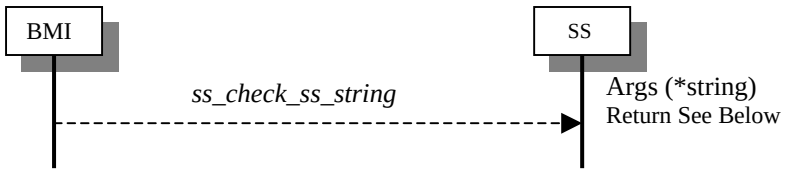
The function *phb_store_entry* is called passing the parameter "book" defined in Read Entries above.

5.12 Supplementary Service

BMI can perform the supplementary service via a menu or from a key sequence directly. In either case a check of the SS string is advisable before the SS call.

5.12.1 Check SS String

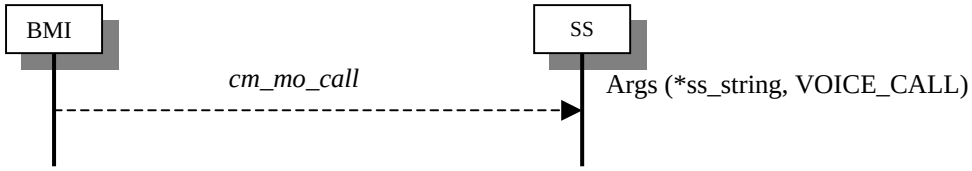
BMI can check an entered string



The function s*s_check_ss_string* is called passing the parameter “string”
The function can return one of the defines in Ref 6.3.1

5.12.2 SS from Key Sequence

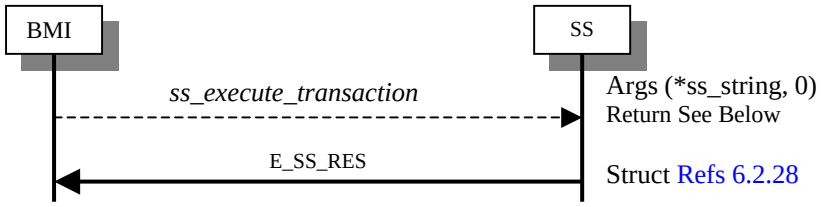
The BMI can perform the supplementary service via a key sequence but only from the idle screen



The function *cm_mo_call* is called passing the following parameters
“ss_string” the key sequence
The function returns one of the defines in Ref 6.3.2

5.12.3 SS from Menu

The BMI can perform the supplementary service via a menu



The function s*s_execute_transaction* is called passing the following parameters
“string” the SS sequence to be performed

The function returns one of the enumerated types defined in Ref 6.3.1
The event E_SS_RES is received indicating that the command has been accepted.
For the following operations a further event confirming the action is received.

5.12.3.1 Call Forwarding

BMI can perform the operations for call forwarding of activation, deactivation, registration, erase and interrogate.
BMI receives the event E_SS_CF_CNF with parameter Structure T_MFW_SS_CF_CNF see Ref 6.2.29

5.12.3.2 Call Waiting

BMI can perform the operations for call waiting of activation, deactivation and interrogation.
BMI receives the event E_SS_CF_CNF with parameter Structure T_MFW_SS_CW_CNF see Ref 6.2.30

5.12.3.3 Call Barring

BMI can perform the operations for call barring of register password, activation, deactivation and interrogation.
For register password operation BMI receives the event E_SS_GET_PW with parameter Structure T_MFW_SS_PW_CNF see Ref 6.2.32
For all other operations it receives the event E_SS_CF_CNF with parameter Structure T_MFW_SS_CB_CNF see Ref 6.2.31

5.12.3.4 Call Line Identification and Connected Line Identification

BMI can perform the operations activate, deactivate and interrogate for the CLIP, CLIR, COLP and COLR
BMI receives the event E_SS_CLI_CNF with parameter Structure T_MFW_SS_CLI_CNF see Ref 6.2.33

5.12.3.5 Un Structured Send USSD

BMI can perform the operation of sending a USSD string
BMI receives the event E_SS_USSD_REQ, with parameter Structure T_MFW_SS_USSD, if the network is waiting for the answer. See Struct Ref 6.2.34
BMI receives the event E_SS_USSD_CNF, with parameter Structure T_MFW_SS_USSD, if the network is not waiting for the answer. See Struct Ref 6.2.34

5.12.3.6 Mobile Equipment Identifier IMEI

BMI can perform the operation of requesting the IMEI number from EEPROM
BMI receives the event E_SS_IMEI with parameter T_MFW_IMEI Structure Ref 6.2.35

5.13 Sim Application Toolkit

The sim application toolkit (SAT) provides The functionality to enables the SIM to access the BMI directly. For example to display startup screens and provide service numbers and menus. The exact nature of the displays depends upon the service provider.
The functions provided by the sim toolkit to handle the sim access are as follows :
Display text
Setup a menu
Select a menu item
Get a user keystroke
Get a user string
Play an audio tone
Setup a call
Get the result of a call
Call control alerting
Send a short message

5.13.1 Initialisation of SAT

The terminal profile detailing the above is sent to the SIM via MFW.

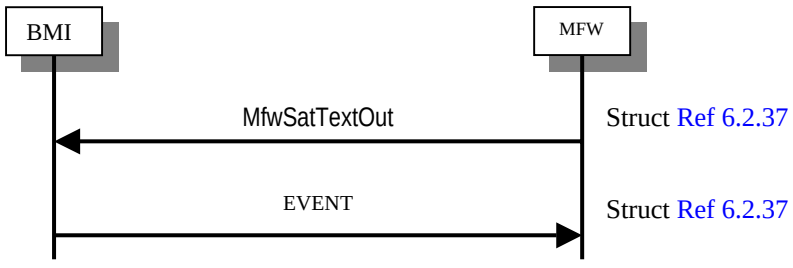
The function s*at_init* is called passing the following parameters
"sat_profile" the sat terminal profile array containing one or more of the following
(SAT_TP1_PRF_DNL | SAT_TP1_MENU_SEL),
(SAT_TP2_CMD_RES | SAT_TP2_CC | SAT_TP2_ALPHA_ID | SAT_TP2_UCS2_ENTRY | SAT_TP2_UCS2_DSPL),
(SAT_TP3_DSPL_TXT | SAT_TP3_GET_INKEY | SAT_TP3_GET_INPUT | SAT_TP3_PLAY_TONE),
(SAT_TP4_SEL_ITEM | SAT_TP4_SEND_SMS | SAT_TP4_SEND_SS | SAT_TP4_SEND_USSD |
SAT_TP4_SETUP_CALL | SAT_TP4_SETUP_MENU),
Every constant refers to a particular SIM Toolkit feature that is supported by BMI.

"size" the size of the array
The SIM card will react differently depending on the terminal profile sent to it. For example, if the SAT_TP4_SETUP_CALL constant was not included in the terminal profile, the BT CELLNET SIM card, which main functionality consists of calls to service numbers, would not try to create the SIM Application Toolkit.

The function returns MfwResOk

5.13.2 Display Text

Sim requests text to be displayed. Callback routine used by BMI to return MFW events to the SIM is *sat_editor_cb*.

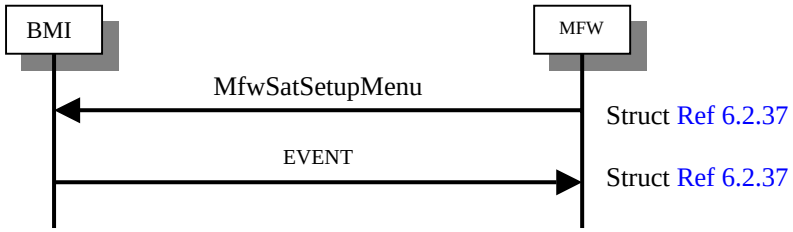


The BMI receives the event MfwSatTextOut requesting display of a text string

The BMI creates the editor and displays the string
On timeout of the display the event SatResSuccess is sent to the SIM
During the display timeout the user can press a key which sends one of the following events to the SIM
SatResSuccess for Ok key
SatResUserHelp for the help key
SatResUserBack for the back key
SatResUserAbort for the abort key
SatResUserNoResp for a timeout

5.13.3 Setup Menus

The SIM accepts the terminal profile and sets up its menus. The Callback routine used by BMI to receive MFW Events to the SIM is *sat_setup_menu_listmnu_cb.*



The BMI receives the event MfwSatSetupMenu to set up the SAT menus
The event contains the label of the menu to be added to the main menu together with a list of items belonging to the SAT main menu, which the user will be able to see after selecting the SAT entry in the main menu.

The BMI processes the command and sends one of the following events to the SIM
SatResSuccess if successfull
SatResImpossible on fail.

5.13.4 Select a Menu Item

An item is selected in the SAT menu to be passed to the SIM. Callback routine used by BMI to return MFW events to the SIM is *sat_select_item_listmnu_cb*.

The BMI receives the event MfwSatSelectItem to request setup of the select item window
The BMI processes the command and sends one of the following events to the SIM
SatResSuccess if successful together with the index of the selected menu
SatResUserBack for the back key
SatResUserAbort for the abort key
SatResUserNoResp for a timeout

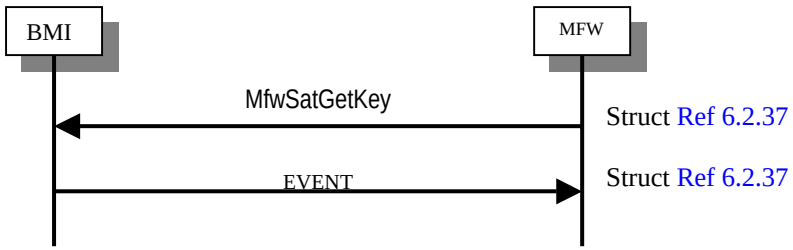
Note: The dynamic list which was scrolled will be destroyed after every selection, being the responsibility of the SIM card to recreate it again when user goes back.

5.13.5 Get Input

Either a key or a string is required.

5.13.5.1 A key

A key is required by the SAT menu to be passed to the SIM. Callback routine used by BMI to return MFW events to the SIM is *sat_editor_cb*.



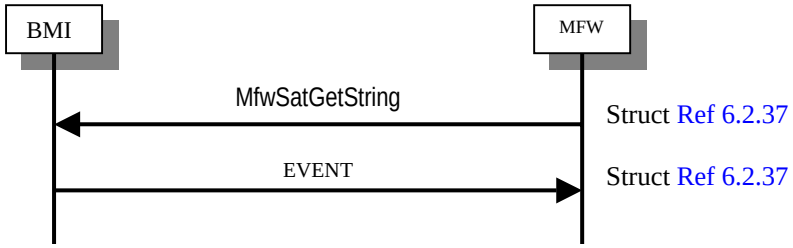
The BMI receives the event MfwSatGetKey requesting an input key

The BMI creates the editor, waits for user input and sends one of the following events to the SIM

SatResSuccess for a single key, 'OK' Key or 'YES' key
SatResUserHelp for the help key
SatResUserBack for the back key
SatResUserAbort for the abort key
SatResUserNoResp for a timeout

5.13.5.2 A String

A string is required by the SAT menu to be passed to the SIM. Callback routine used by BMI to return MFW events to the SIM is *sat_editor_cb*.

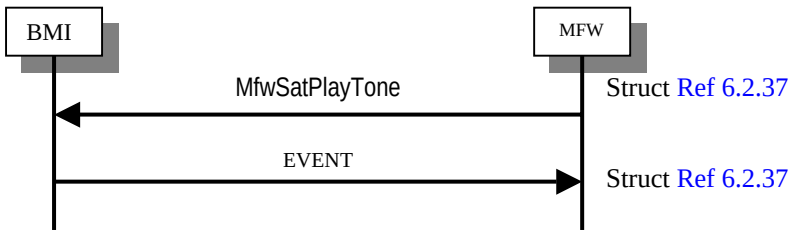


The BMI receives the event MfwSatGetString requesting an input string
The BMI creates the editor, waits for user input and sends one of the following events to the SIM
SatResSuccess for a string followed by the 'OK' key

Note: A number of string conversion are necessary in this process, because the type used by SAT(UC2) and BMI (ASCII) are not the same

5.13.6 Play a Tone

The SIM request playing of a tone. The Callback routine used by BMI to receive MFW Events is *sat_play_tone_tim_cb*

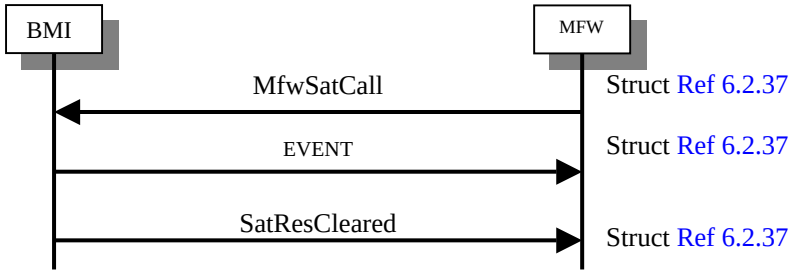


The BMI receives the event SatResPlayTone to play the requested tone

The BMI processes the command and sends one of the following events to the SIM
SatResSuccess as the tone has timed out
SatResAbort as the user has aborted the tone.

5.13.7 Setup a Call

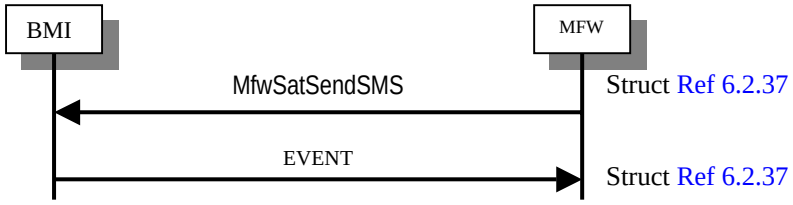
The SIM request a call to be set up. The Callback routine used by BMI to receive MFW Events is *sat_call_setup_cb*.



The BMI receives the event MfwSatCall to setup a call
The BMI waits for the user to accept the call and sends one of the following events to the SIM
SatResSuccess if the user accepts
SatResReject if the user rejects the call
When the call ends the BMI sends the event SatResCleared

5.13.8 SMS

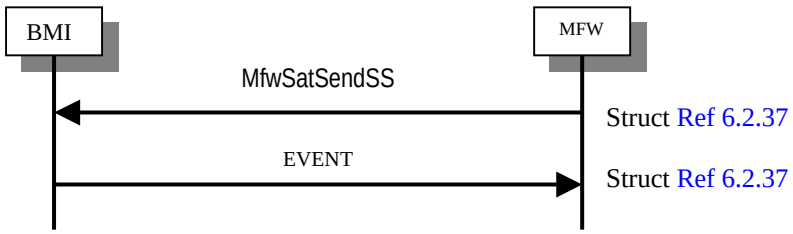
The SIM requests an SMS string. The Callback routine used by BMI to receive MFW Events is



The BMI receives the event MfwSatSendSMS for a short message service string
The BMI processes the command and sends one of the following events to the SIM
SatResSuccess as the SMS string is available

5.13.9 SS

The SIM requests an SS string. The Callback routine used by BMI to receive MFW Events is



The BMI receives the event MfwSatSendSS for a supplementary service string
The BMI processes the command and sends one of the following events to the SIM
SatResSuccess as the SS string is available

5.13.10 Ending the SAT session

The BMI receives the event MfwSatSessionEnd to end the current SAT session.

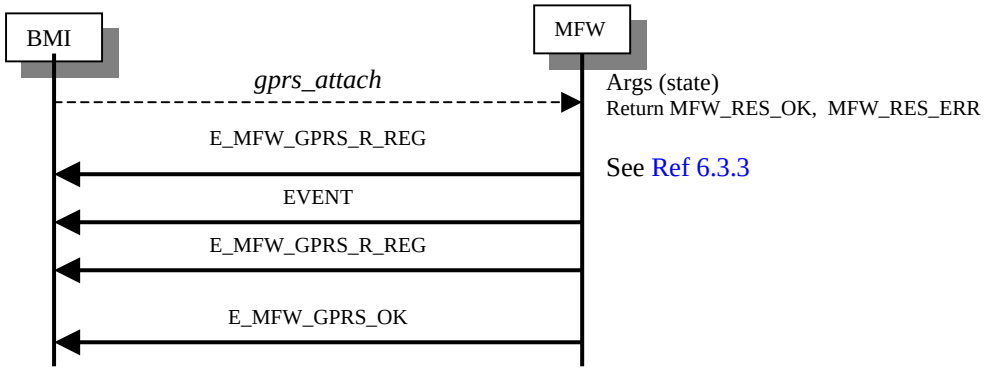
5.14 GPRS

GPRS is a general packet service with data rates up to 21.4 Kbps and eight time slots. Callback routine used by BMI to receive MFW Events is *GPRSMfwCb*.

5.14.1 Network Attachment

The BMI attaches or detaches from the GPRS service.

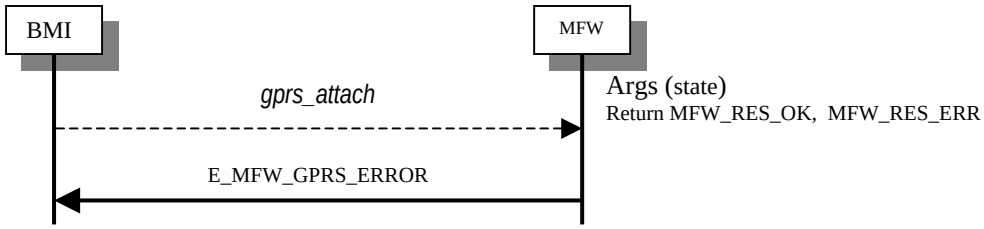
5.14.1.1 Attach / Detach Successful



The function *gprs_attach* is called passing the 'state' parameter which can have the following values
CGATT_STATE_ATTACHED
CGATT_STATE_DETACHED

The event E_MFW_GPRS_R_REG is received indicating that the command has started
Followed by the event E_MFW_GPRS_S_ATT for successful attachment or E_MFW_GPRS_R_EREP_ATT for successful detachment then the event E_MFW_GPRS_R_REG again
Finally the event E_MFW_GPPRS_OK is received

5.14.1.2 Attach / Detach Fails



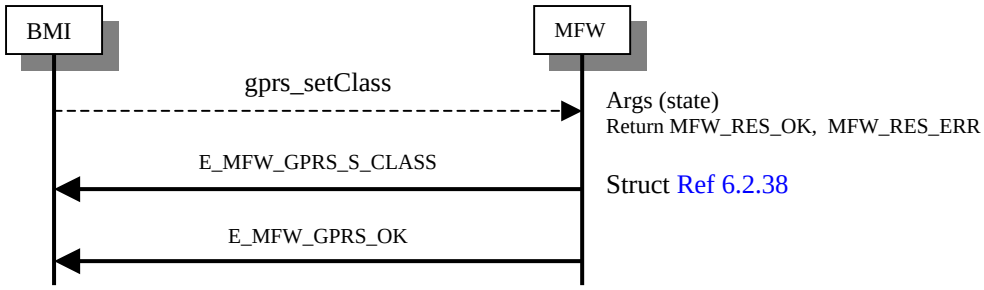
The function *gprs_attach* is called

The event E_MFW_GPRS_ERROR is received indicating that the command has failed

5.14.2 Set Mobile Class

The BMI sets the GPRS mobile class.

5.14.2.1 Set Class Successful

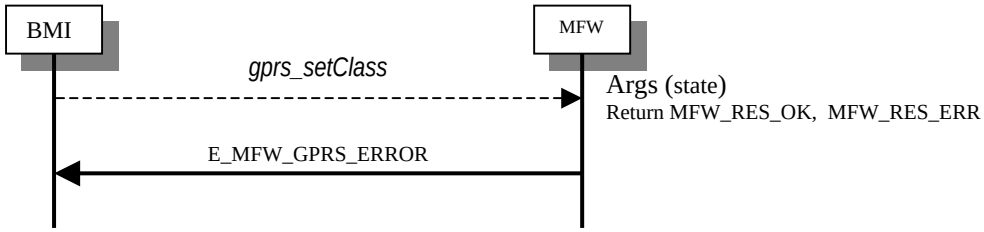


The function *gprs_setClass* is called passing the 'class' parameter which can have one of the following values

CGCLASS_CLASS_OMITTED	value is omitted
CGCLASS_CLASS_A	mobile class A (highest)
CGCLASS_CLASS_B	mobile class B (if necessary consider NET III)
CGCLASS_CLASS_C	mobile class C (circuit switched)
CGCLASS_CLASS_CG	mobile class CG (GPRS only mode)
CGCLASS_CLASS_CC	mobile class CC (circuit switched only mode - lowest)
CGCLASS_CLASS_MAX	invalid value

The event E_MFW_GPRS_R_EREP_CLASS is received indicating that the command has started
Followed by the event E_MFW_GPRS_S_CLASS
Finally the event E_MFW_GPPRS_OK is received

5.14.2.2 Set Class Fails



The function *gprs_attach* is called

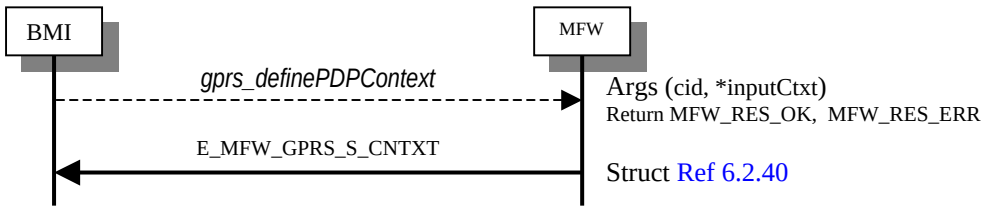
The event E_MFW_GPRS_ERROR is received indicating that the command has failed

5.14.3 PDP

The BMI can set the various parameters associated with the PDP service

5.14.3.1 Set PDP Context

The BMI sets the GPRS PDP context.

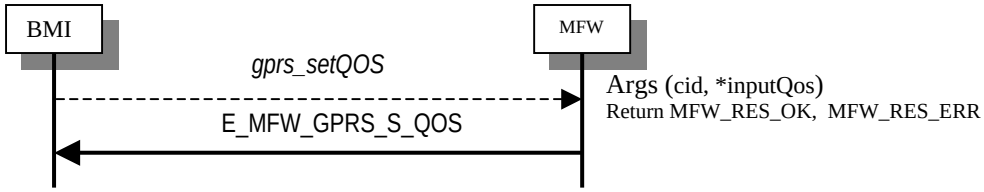


The function *gprs_definePDPContext* is called passing the following parameters
context id 'cid'
pointer to a context record 'inputCtxt' see Ref 6.2.39

The event E_MFW_GPRS_CNTXT is received indicating that the command has been successful

5.14.3.2 Set PDP Quality of Service

The BMI sets the GPRS PDP quality of service.

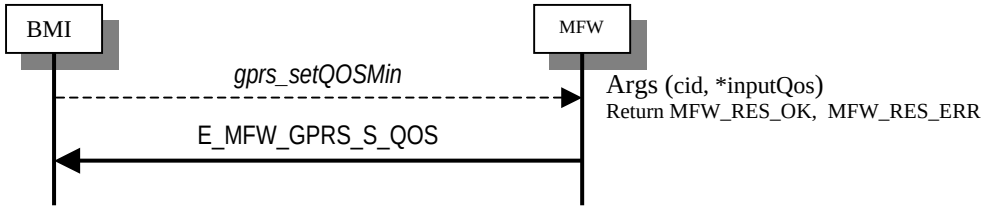


The function *gprs_setQOS* is called passing the following parameters
quality of service 'inputQos'
pointer to a quality of service record 'inputQos' see Ref 6.2.39.1

If successful MFW_RES_OK is returned else MFW_RES_ERR followed by event E_MFW_GPRS_S_QOS.

5.14.3.3 Set PDP Minimum Quality of Service

The BMI sets the GPRS minimum PDP quality of service.



The function *gprs_setQOSMIn* is called passing the following parameters
context id 'cid'
pointer to a quality of service record 'inputQos' see Ref 6.2.39.1

If successful MFW_RES_OK is returned else MFW_RES_ERR followed by event E_MFW-GPRS_S_QOS_MIN.

5.14.3.4 Asynchronous PDP Minimum Quality of Service Change

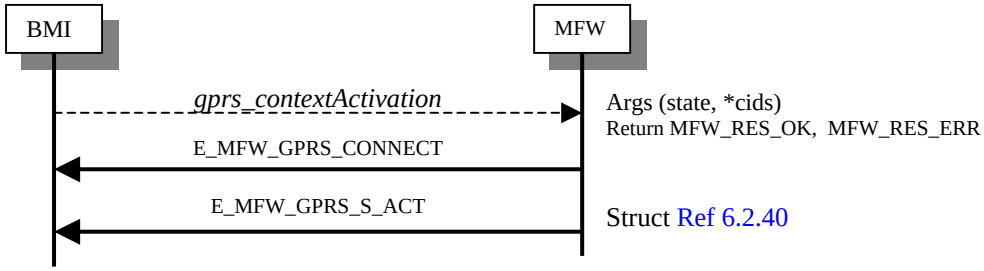
The quality of service can change at any time and when this happens the BMI is informed.

The event E_MFW_GPRES_R_QOS is sent to the BMI

5.14.3.5 PDP Context Activation

The BMI can activate or deactivate the GPRS PDP context.

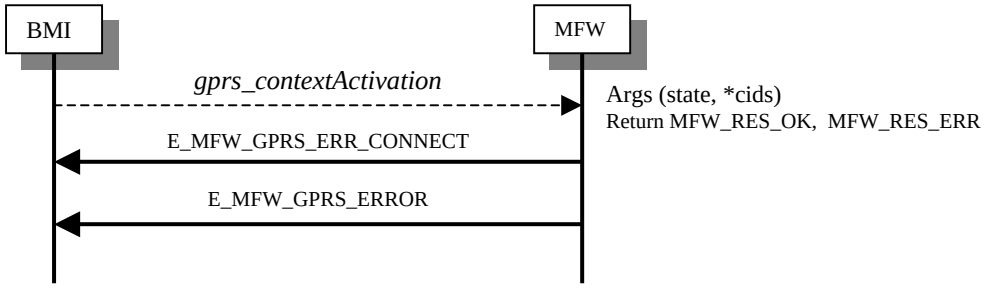
5.14.3.5.1 Activation Successful



The function *gprs_contextActivation* is called passing the following parameters
the 'state' which can have the following values
CGACT_STATE_OMITTED value is omitted
CGACT_STATE_DEACTIVATED PDP context detached
CGACT_STATE_ACTIVATED PDP context attached
CGACT_STATE_INVALID invalid value
pointer to a list of context definitions 'cids'

The event E_MFW_GPRS_CONNECT is received indicating that the command has been successful
Followed by the event E_MFW_GPRS_R_ACT giving context activation report

5.14.3.5.2 Activation Fails

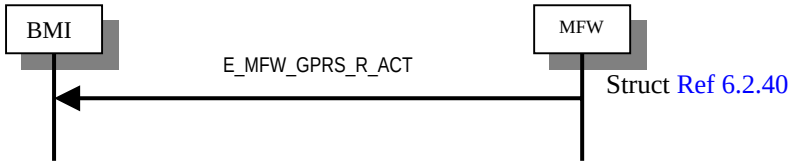


The function *gprs_contextActivation* is called

The event E_MFW_GPRS_ERR_CONNECT is received indicating that the command has failed
Followed by the event E_MFW_GPRS_ERROR

5.14.3.6 Asynchronous PDP Context Activation

The context activation state can change at any time and when this happens the BMI is informed.

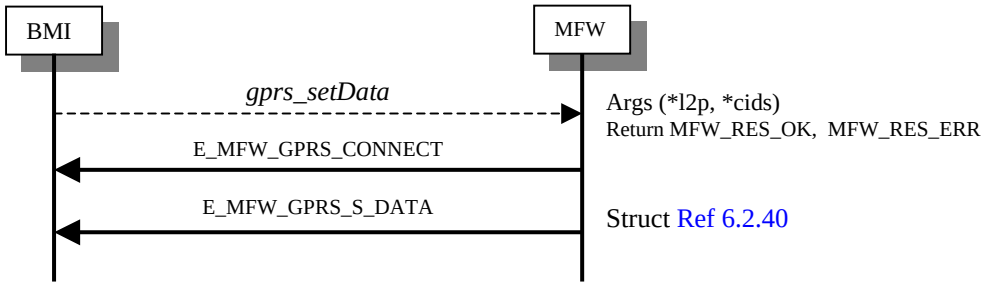


The event E_MFW_GPRS_R_ACT is sent to the BMI

5.14.3.7 PDP Data State

The BMI can establish communication with the network GPRS PDP context.

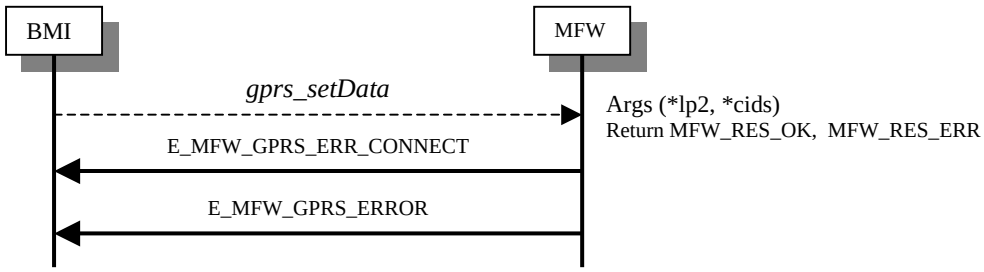
5.14.3.7.1 Communication Successful



The function *gprs_setData* is called passing the following parameters
'lp2' a pointer to one of the layer 2 protocols
'cids' pointer to a list of context definitions

The event E_MFW_GPRS_CONNECT is received indicating that the command has been successful
Followed by the event E_MFW_GPRS_R_DATA giving context activation report

5.14.3.7.2 Communication Fails

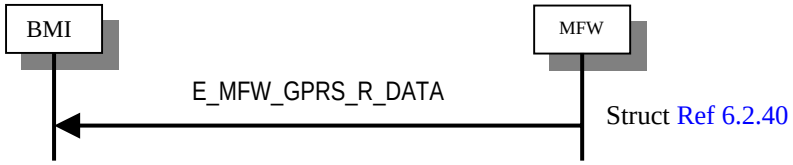


The function *gprs_setData* is called

The event E_MFW_GPRS_ERR_CONNECT is received indicating that the command has failed
Followed by the event E_MFW_GPRS_ERROR

5.14.3.7.3 Asynchronous PDP Communication

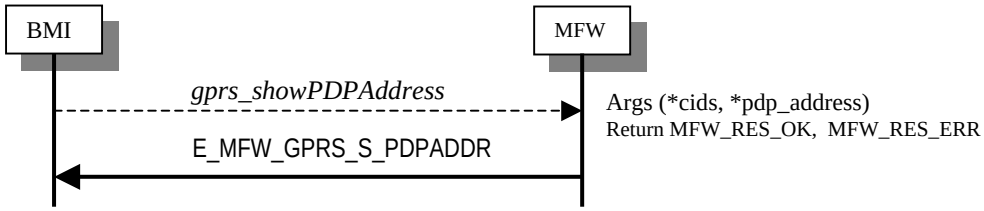
The communication state can change at any time and when this happens the BMI is informed.



The event E_MFW_GPRS_R_DATA is sent to the BMI

5.14.3.8 PDP Address

The BMI can set the GPRS PDP address.

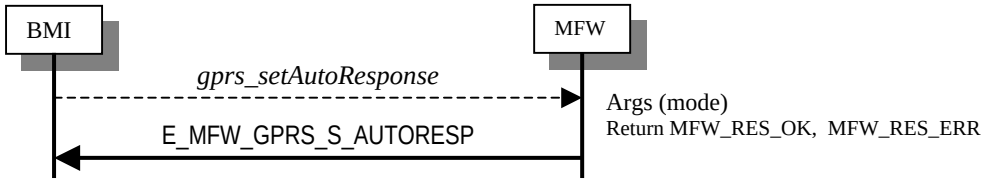


The function *gprs_showPDPAddress* is called passing the following parameters
pointer to a list of context definitions ‘cids’
pointer to a pdp address

The function returns MFW_RES_OK if successful else MFW_RES_ERR

5.14.3.9 PDP Auto Response Mode

The BMI can enable or disable automatic response to the receipt of context activation requests from the network.

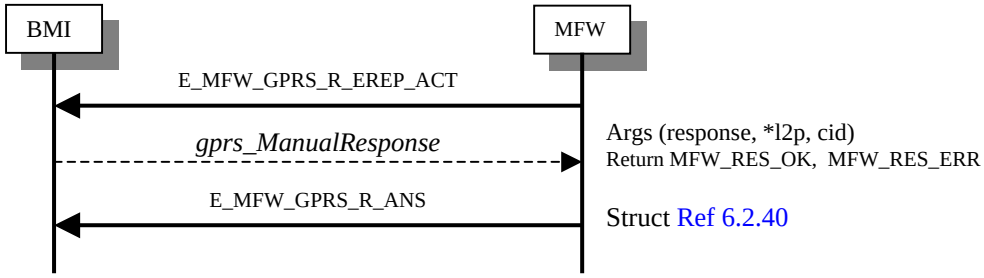


The function *gprs_setAutoResponse* is called passing 'mode' which can have one of the following values

CGAUTO_N_OMITTED	value is omitted
CGAUTO_N_GPRS_RES_OFF	turn off automatic response
CGAUTO_N_GPRS_RES_ON	turn on automatic response
CGAUTO_N_MCM_GPRS_ONLY	modem compatibility mode
CGAUTO_N_MCM_GPRS_CSC	modem compatibility mode, GPRS and circuit switched calls
CGAUTO_N_INVALID	invalid value

5.14.3.10 PDP Manual Mode

The BMI can accept or reject a response from the network for context activation



BMI receives the event E_MFW_GPRS_R_EREP_ACT requesting a manual response to the network request
The function *gprs_ManualResponse* is called passing the following parameters
'response' which can have one of the following values

CGANS_RESPONSE_OMITTED	value is omitted
CGANS_RESPONSE_REJECT	reject the request
CGANS_RESPONSE_ACCEPT	accept and request that the PDP context be activated

a pointer to one of the layer 2 protocols 'lp2'
context definitions 'cid'

The event E_MFW_GPRS_R_ANS is received indicating the command has been accepted
Followed by the event E_MFW_GPRS_CONNECT giving connect establishment
and finally E_MFW_GPRS_R_DATA for data state reporting.

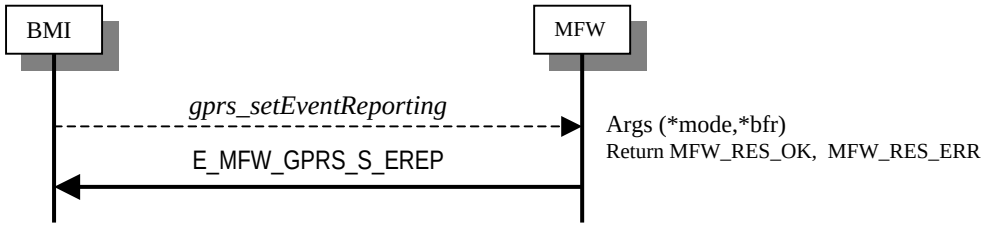
5.14.3.10.1 Asynchronous Manual Answer Reporting

The manual answer reporting can be sent to the BMI at any time.

The event E_MFW_GPRS_R_ANS is sent to the BMI

5.14.3.10.2 Event Reporting

The BMI can enable or disable the sending of certain events occuring in the GPRS network



BMI call The function *gprs_setEventReporting* is called passing the following parameters
'mode' which can have one of the following values

CGEREP_MODE_OMITTED	value is omitted
CGEREP_MODE_BUFFER	buffer unsolicited result codes
CGEREP_MODE_DICARD_RESERVED	discard unsolicited result codes when MT-TE link is reserved
CGEREP_MODE_BUFFER_RESERVED	buffer unsolicited result codes when MT-TE link is reserved
CGEREP_MODE_INVALID	invalid value

'bf'r' which is a pointer to a the buffered events which can have one of the following values

CGEREP_BFR_OMITTED	value is omitted
CGEREP_BFR_CLEAR	buffer unsolicited result codes
CGEREP_BFR_FLUSH	discard unsolicited result codes when MT-TE link is reserved
CGEREP_BFR_INVALID	invalid value

The function returns MFW_RES_OK if successful else MFW_RES_ERR

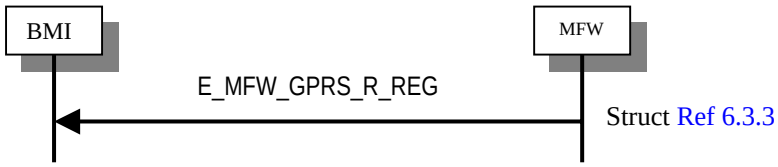
5.14.3.10.3 Asynchronous Event Reporting

A number of events relating to reporting can be sent to the BMI at any time.

BMI receives the event E_MFW_GPRS_R_EREP_RJ to indicate rejection of event reporting see
or event E_MFW_GPRS_R_EREP_DEACT to indicate deactivation of event reporting
or event E_MFW_GPRS_R_EREP_CLASS to indicate mobile class change event reporting

5.14.3.10.4 Asynchronous Registration Status

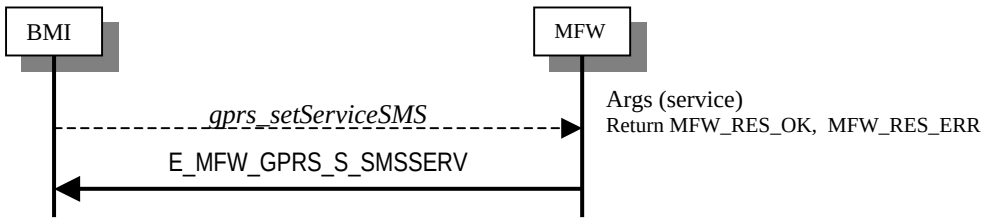
The registration status can be sent to the BMI at any time.



BMI receives the event E_MFW_GPRS_R_REG to inform of changing registration status

5.14.3.11 SMS Service

The BMI can set the service preference for SMS



BMI calls The function *gprs_setServiceSMS* is called passing the parameter 'service' which can have one of the following values

CGSMS_SERVICE_OMITTED	value is omitted
CGSMS_SERVICE_GPRS	GPRS
CGSMS_SERVICE_CS	circuit switched
CGSMS_SERVICE_GPRS_PREFERRED	GPRS preferred
CGSMS_SERVICE_CS_PREFERRED	circuit switched preferred
CGSMS_SERVICE_INVALID	invalid value

The function returns MFW_RES_OK if successful else MFW_RES_ERR

5.15 WAP

The Wireless Application Protocol uses the windows features of MFW to set up menus and editors. It bypasses the MFW to communicate with the protocol stack directly.

6 Appendix

6.1 MFW Window Structures

6.1.1 Definition of MfwHnd

This defines the MFW element handle.

void * MfwHnd

6.1.2 Structure MfwWinAttr

This Structure defines the window attributes.

MfwRect	win	window position and size
MfwRect	view	view position and size
U16	bgColor	background color

struct MfwRect

U16	px	top left pos x
U16	py	top left pos y
U16	sx	horizontal size
U16	sy	vertical size

6.1.3 Structure MfwWin

This Structure defines the window control block.

MfwEvt	mask	selection of events
MfwEvt	flags	current event
MfwCb	handler	event handler
MfwWinAttr	*attr	window attributes
MfwHdr	*elems	window elements
void	*user	user specific data

struct MfwHdr

MfwHdr	*next	next header
MfwTyp	type	elem type code
void	*data	elem control Structure

6.2 Event Structures

6.2.1 Structure T_MFW_SIM_STATUS

The Structure are used during the SIM activation phase and pin changes.

6.2.1.1 No Pin required

char sim_status	= MFW_SIM_NO_PIN
char sim_operation_mode	= sim_config.oper_mode
char sim_pin_retries	= 0
char sim_procedure	= MFW_SIM_ACTIVATION
char sim_status_type	= MFW_SIM_PIN1

6.2.1.2 Pin required

sim_status	= MFW_SIM_PIN_REQ
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= pin retries count
sim_procedure	= MFW_SIM_ACTIVATION
sim_status_type	= MFW_SIM_PIN1 or MFW_SIM_PIN2

6.2.1.3 Correct Pin

sim_status	= MFW_SIM_NO_PIN
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= 0
sim_procedure	= MFW_SIM_VERIFY
sim_status_type	= MFW_SIM_PIN1 or MFW_SIM_PIN2

6.2.1.4 Incorrect Pin

sim_status	= MFW_SIM_PIN_REQ
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= pin retries count
sim_procedure	= MFW_SIM_VERIFY
sim_status_type	= MFW_SIM_PIN1 or MFW_SIM_PIN2

6.2.1.5 PUK Required

sim_status	= MFW_SIM_PUK_REQ
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= puk1 retries count
sim_procedure	= MFW_SIM_ACTIVATION or MFW_SIM_VERIFY
sim_status_type	= MFW_SIM_PUK1 or MFW_SIM_PUK2

6.2.1.6 Correct PUK

sim_status	= MFW_SIM_SUCCESS
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= 0
sim_procedure	= MFW_SIM_UNBLOCK
sim_status_type	= MFW_SIM_PUK1 or MFW_SIM_PUK2

6.2.1.7 Incorrect PUK

sim_status	= MFW_SIM_PUK_REQ
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= pin retries count
sim_procedure	= MFW_SIM_UNBLOCK
sim_status_type	= MFW_SIM_PUK1 or MFW_SIM_PUK2

6.2.1.8 SIM Card Unavailable

sim_status	= MFW_SIM_INVALID_CARD or MFW_SIM_NO_SIM_CARD
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= 0
sim_procedure	= MFW_SIM_ACTIVATION
sim_status_type	= MFW_SIM_UNKNOWN

6.2.1.9 Pin Change correct

sim_status	= MFW_SIM_SUCCESS
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= 0
sim_procedure	= MFW_SIM_CHANGE
sim_status_type	= MFW_SIM_PIN1or MFW_SIM_PIN2

6.2.1.10 Pin Change incorrect

sim_status	= MFW_SIM_FAILURE
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= 0
sim_procedure	= MFW_SIM_CHANGE
sim_status_type	= MFW_SIM_PIN1or MFW_SIM_PIN2

6.2.1.11 Pin Enable correct

.sim_status	= MFW_SIM_SUCCESS
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= 0 =
sim_procedure	= MFW_SIM_ENABLE
sim_status_type	= MFW_SIM_UNKNOWN

6.2.1.12 Pin Enable incorrect

sim_status	= MFW_SIM_FAILURE
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= retries count
sim_procedure	= MFW_SIM_ENABLE
sim_status_type	= MFW_SIM_UNKNOWN

6.2.1.13 Pin Disable correct

The Structure is used when disabling the pin.

sim_status	= MFW_SIM_SUCCESS
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= 0
sim_procedure	= MFW_SIM_DISABLE
sim_status_type	= MFW_SIM_UNKNOWN

6.2.1.14 Pin Disable incorrect

The Structure is used when disabling the pin.

sim_status	= MFW_SIM_FAILURE
sim_operation_mode	= sim_config.oper_mode
sim_pin_retries	= retries count
sim_procedure	= MFW_SIM_DISABLE
sim_status_type	= MFW_SIM_UNKNOWN

6.2.2 Structure T_MFW_CM_REDIAL

The Structure are used during either manual or automatic redial.

6.2.2.1 Redial started

char redial_mode	= CM_REDIAL_STARTED

short call_id = not used
char number [ ] = string of the call number
char name [ ] = string of the alphabet name
char subaddr [ ] = not used
char ton = type of the number
char type = type of this call (indicates the voice call, data call ...)
char left_attempts = not used

6.2.2.2 Redial attempt

redial_mode = CM_REDIAL_ATTEMPT
call_id = call id
number = string of the call number
name = string of the alphabet name
subaddr = not used
ton = type of the number
type = type of this call (indicates the voice call, data call ...)
left_attempts = left attempts

6.2.2.3 Redial success

redial_mode = CM_REDIAL_SUCCESS
call_id = not used
number = not used
name = not used
subaddr = not used
ton = not used
type = not used

6.2.2.4 Redial next attempt

redial_mode = CM_REDIAL_ATTEMPT
call_id = call id
number = string of the call number
name = string of the alphabet name
subaddr = not used
ton = type of the number
type = type of this call (indicates the voice call, data call ...)
left_attempts = left attempts

6.2.2.5 Redial blacklisted

redial_mode = CM_REDIAL_BLACKLISTED
call_id = not_used
number = string of the call number
name = string of the alphabet name
subaddr = not used
ton = type of the number
type = type of this call (indicates the voice call, data call ...)
left_attempts = 0

6.2.2.6 Redial manual

redial_mode = CM_REDIAL_MANU
call_id = not used
number = string of the call number
name = string of the alphabet name
subaddr = not used
ton = type of the number
type = type of this call (indicates the voice call, data call ...)
left_attempts = left attempts

6.2.3 Structure T_MFW_PLMN_IDENT

The Structure is used during network registration to give operator details
char network_long [ ] = operator name in long format
char network_short [ ] = operator name in short format
char network_numeric [ ] = operator name in numeric format
char service_provider = service provider name
char roaming_indicator = PLMN in HPLMN or not.

6.2.4 Structure T_MFW_PLMN_LIST

The Structure is used during network registration to give a list of available networks
char result = result of PLMN search
char count = number of available PLMNs
T_PLMN plmn [ ] = plmn identifications

6.2.5 Structure T_MFW_PLMN

The Structure is used during network registration to give operator details of each network
char network_long [ ] = operator name in long format
char network_short [ ] = operator name in short format
char network_numeric [ ] = operator name in numeric format
char roaming_indicator = PLMN is HPLMN or not.
char forbidden_indicator = PLMN is member of the forbidden PLMN list
char *fieldstrength = fieldstrength of PLMN

6.2.6 Structure T_MFW_PREF_PLMN_LIST

The Structure is used to give a list of available networks
char count = number of entries in preferred PLMN list
T_PREF_PLMN *plmn = plmn identifications

6.2.6.1 Structure T_PREF_PLMN

The Structure is used to give a list of plmn identifications
short Index = physical position in SIM card
byte network_long [ ] = operator name in long format
byte network_short [ ] = operator name in short format
byte network_numeric [ ] = operator name in numeric format

6.2.7 Structure T_MFW_PPLMN_MEM

The Structure is used to give a list of plmn identifications
char MaxRc = maximum preferred PLMN records in SIM card
char usedRcd = used preferred PLMN records in SIM card

6.2.8 Structure T_MFW_PREF_PLMN

The Structure is used to give the home plmn
short index = not used
char network_long [ ] = operator name in long format
char network_short [ ] = operator name in short format
char network_numeric [ ] = operator name in numeric format

6.2.9 Structure T_MFW_SIM_PIN_STATUS

char type = MFW_SIM_PIN1
MFW_SIM_PIN2

char set = MFW_SIM_ENABLE
MFW_SIM_DISABLE

char stat = MFW_SIM_NO_PIN
MFW_SIM_PIN_REQ
MFW_SIM_PIN2_REQ

6.2.10 Structure T_MFW_CM_CW_INFO

The Structure gives information on an incoming call

char result = Present
short call_number = call id
char number [ ] = the number of the called party
char ton = type of number

char type = VOICE_CALL
AuxVOICE_CALL,
DATA_CALL,
FAX_CALL,
VFD_VOICE voice followed data, voice mode
VAD_VOICE voice alternating data, voice mode
VAF_VOICE voice alternating fax, voice mode
VFD_DATA voice followed data, data mode
VAD_DATA voice alternating data, data mode
VAF_FAX voice alternating fax, fax mode

char name [ ] = alpha identfier
char subaddr [ ] = not used
char ri = not used
T_MFW_CM_BC_PARA bc1 = not used for voice call
T_MFW_CM_BC_PARA bc2 = not used for voice call

Note: “number” is filled when the network delivers the called party number
“name” is filled when the corresponding alpha identifier exists in the phonebook.

6.2.11 Structure T_MFW_CM_MO_INFO

The Structure is the used to start an outgoing call

SHORT call_id = call identify
UBYTE number [ ] = call number
UBYTE ton = type of number
UBYTE type
T_MFW_PHB_TEXT name = alpha identfier
UBYTE subaddr [ ] = subaddres

6.2.12 Structure T_MFW_CM_CPI

The Structure is the call progress indication gives information on the audio status of a call

short call_number = call id
T_MFW_CPI_TYPE type = CPI_TYPE_ALERT
T_MFW_CPI_TYPE inband = CPI_IBT_TRUE
T_MFW_CPI_TYPE tch = CPI_TCH_TRUE

6.2.13 Structure T_MFW_CM_DISCONNECT

The Structure gives information on a disconnected call

SHORT call_number = call number
UBYTE cause = reason for disconnect

6.2.14 Structure T_MFW_CM_COMMAND

The Structure is the call commands

short call_number = number of the call

byte command	= CM_HOLD CM_RETRIEVE CM_SWAP CM_BUILD_MULTIPARTY CM_HOLD_MULTIPARTY CM_RETRIEVE_MULTIPARTY CM_SPLIT_MUTLIPARTY CM_ETC
byte result	= Present or NotPresent

6.2.15 Structure T_MFW_CM_AOC_CNF.

The Structure are used when setting a parameter for advice of charge.

6.2.15.1 Setting of AoC parameter correct

byte requested_type	= request_type
T_MFW_AOC_CNF_RES result	= CM_AOC_CONF_OK

6.2.15.2 Setting of AoC parameter fails

requested_type	= request_type
result	= CM_AOC_CONF_UNKNOWN CM_AOC_CONF_SIMPIN2REQ CM_AOC_CONF_WRONGPASSWD CM_AOC_CONF_OPNOTALLOW CM_AOC_CONF_ACMRESETNEEDED

6.2.16 Structure T_MFW_CM_AOC_INFO.

The Structure is used when getting a parameter for advice of charge.

byte requested_type	= requested type
long actual_call_timer	= time of actual call
long ccm	= current call meter
byte ccm_puct [ ]	= current call meter using PUCT
long acm	= accumulated call meter
byte acm_puct [ ]	= accumulated call meter using PUCT
long acm_max	= accumulated call meter maximum PUCT
byte cur [ ]	= currency table
long eppu	= elementary price per unit
long exp	= decimal logarithm
long sexp	= sign of decimal logarithm
byte ppu [ ]	= price per unit

6.2.17 Structure T_MFW_SMS_ IDX

The Structure is used to store the index of the SMS message list.

short Index	= message index
T_MFW_SMS_STAT stat	
T_MFW_SMS_STAT	

6.2.18 Structure T_MFW_SMS_MSG

The Structure is used to store the parameters of the SMS message list.

short Index	= message index
byte msg_ref	= message reference
T_MFW_SMS_STAT stat	= status of sms message
T_MFW_SMS_STAT msg_type	= voice mail status
T_MFW_SMS_STAT addr	= called party address
T_MFW_SMS_STAT rctp	= received sms timestamp

6.2.19 Structure T_MFW_SMS_MT

The Structure is used to store a mobile terminated SMS message.

byte Index	= index of message
T_MFW_SMS_ADDR orig_addr	= call party address
char sc_addr [ ]	= service centre address
char prot_id	= protocol identifier
short dcs	= data coding scheme
char rp	= reply path
T_MFW_SMS_SCTP sctp	= service centre timestamp
char msg_len	= length of short message
char sms_msg [ ]	= short message
char udh_len	= length of user data header
char udh [ ]	= user data header

6.2.19.1 Structure T_MFW_SMS_ADDR

char tag [ ]	= mfw tag name
char number [ ]	= number
T_MFW_PHB_TON ton	= type of number
T_MFW_PHB_NPI npi	= numbering plan identifier

6.2.19.2 Structure T_MFW_SMS_SCTP

char year [ ]	= Two digit date information
char month [ ]	= “”
char day [ ]	= “”
char hour [ ]	= “”
char minute [ ]	= “”
char second [ ]	= “”
short timezone	=

6.2.20 Structure T_MFW_SMS_CB

The Structure is used to store a cell broadcast SMS message.

short Sn	= serial number in integer format
char msg_id	= message identification
char dcs	= data coding scheme
char page	= number of this page
char pages	= number of total pages
char msg_len	= length of short message
char cb_msg [ ]	= cell broadcast message

6.2.21 Structure T_MFW_SMS_INFO

The Structure is used to store the SMS information parameters

char sc_addr [ ]	= service centre address
short prot_id	= protocol identifier
short dcs	= data coding scheme
char srr	= status report request
char rp	= reply path
char rd	= reject duplicates
char vp_mode	= period mode
char vp_rel	= validity period relative
T_MFW_SMS_SCTP vp_abs	= validity period absolute

6.2.22 Structure T_MFW_SMS_MO

The Structure is used for a mobile originated message

T_MFW_SMS_ADDR dest_addr	= called party address
CHAR sc_addr [ ]	= service centre address
UBYTE prot_id	= protocol identifier

SHORT dcs	= data coding scheme
UBYTE status	= status of sms message
UBYTE vp_mode	= validity period mode
UBYTE vp_rel	= validity period relative
T_MFW_SMS_SCTP vp_abs	= validity period absolute
CHAR sms_msg [ ]	= short message
UBYTE msg_len	= length of short message
UBYTE msg_ref	= message reference

6.2.23 Structure T_MFW_SMS_ID

The Structure is used for message indication both MT and CB

UBYTE mem	= indicate the memory for stored message
UBYTE index	= index of message

6.2.24 Structure tMmiPhbData

The Structure is used to store the result of a read or search

T_MFW_PHB_ENTRY entry [ ]	=
T_MFW_PHB_LIST list	= list of entries
T_MFW_PHB_STATUS status	= info on the phonebook
UBYTE result	= result of the search
UBYTE selectedName	=
UBYTE index	= start of entries to search
int mode	= read order
tSearchTypes KindOfSearch	= SEARCH_BY_NAME SEARCH_BY_NUMBER SEARCH_BY_LOCATION

6.2.25 Structure T_MFW_PHB_LIST

The Structure is used to store a list of phonebook entries

byte result	= result of operation
byte book	= phonebook
byte num_entries	= number of phonebook entries
T_MFW_PHB_ENTRY *entry	= phonebook entries

6.2.26 Structure T_MFW_PHB_ENTRY

The Structure is used to store a phonebook entry

byte book	= phonebook
char [ ]	= alpha identifier if Ascii
char name [ ]	= else length of name
char number [ ]	= telephone number
char index	= index in phonebook table
char ton	= type of number
char npi	= numbering plan identifier
T_MFW_PHB_DATE date	= date of entry
T_MFW_PHB_TIME time	= time of entry

6.2.26.1 Structure T_MFW_PHB_DATE

char year [ ]	= Two digit date info
char month [ ]	= “”
char day [ ]	= “”

6.2.26.2 Structure T_MFW_PHB_TIME

char hour [ ]	= Two digit time info
char minute [ ]	= “”
char second [ ]	= “”

6.2.27 Structure T_MFW_PHB_STATUS

The Structure is used to provide the status of a phonebook

char book	= phonebook
char max_entries	= max entries in the phonebook
char tag_len	= length of alpha identifier
char used_entries	= used entries in the phonebook
char avail_entries	= entries available

6.2.28 Structure T_MFW_SS_RES

The Structure is used to provide the context of a SS command

char type	= type of service
char category	= command category
char ppn [ ]	= password, PUK, number depending on SS type
char pwd [ ]	= password if applicable
char bs	= basic service
char time	= no reply condition time

6.2.29 Structure T_MFW_SS_CF_CNF

The Structure is used during call forwarding

enum ss_code	= SS_CF_ALL	call call forwarding
	SS_CF_ALL_COND	call conditional call forwarding
	SS_CF_CFU	call forwarding unconditional
	SS_CF_CFNRY	call forwarding no reply
	SS_CF_CFNR	call forwarding not reachable
	SS_CF_CFB	call forwarding busy

enum ss_category = SS_REGISTRATION
SS_ERASURE
SS_ACTIVATION
SS_DEACTIVATION
SS_INTERROGATION

enum ss_error = MFW_SS_NO_ERROR
MFW_SS_ERROR

char ss_feature_count	= Number of feature elements
T_MFW_FEATURE *ss_feature_list	= Feature list

6.2.29.1 Structure T_MFW_FEATURE

char ss_telecom_type
char ss_telecom_service
char ss_status
char ss_numbering_type
char ss_numbering_plan
char ss_forwarded_to_number [ ]
char ss_forwarded_to_subaddress [ ]
char ss_no_reply_condition_time

6.2.30 Structure T_MFW_SS_CW_CNF

The Structure is used during call waiting

enum ss_code = SS_CW

enum ss_category = SS_ACTIVATION
SS_DEACTIVATION
SS_INTERROGATION

enum ss_error	= MFW_SS_NO_ERROR	
	MFW_SS_ERROR	
char ss_status	= bitmap of SS status:	
	SS_STAT_ACTIVE	0x01
	SS_STAT_REGISTERED	0x02
	SS_STAT_PROVISIONED	0x04
	SS_STAT_QUIESCENT	0x08
	SS_STAT_UNKNOWN	0xFF
char ss_service_count	= Number of telecom elements	
T_MFW_SRV_GROUP *ss_service_list	= Service list	

6.2.30.1 Structure T_MFW_SRV_GROUP

char ss_telecom_type
char ss_telecom_service

6.2.31 Structure T_MFW_SS_CB_CNF

The Structure is used during call barring

enum ss_code	=	SS_CB_BAOC	barr outgoing calls
		SS_CB_BAOIC	bar outgoing international calls
		SS_CB_BAOICexH	bar out international calls except HPLMN
		SS_CB_BAIC	barr of incoming calls
		SS_CB_BAICroam	barr of incoming calls when roaming
		SS_CB_ALL	all call barring
		SS_CB_ALL_BAOC	all outgoing call barring
		SS_CB_ALL_BAI	all incoming call barring
enum ss_category	=	SS_ACTIVATION	
		SS_DEACTIVATION	
		SS_INTERROGATION	
enum ss_error	=	MFW_SS_NO_ERROR	
		MFW_SS_ERROR	
char ss_telecom_count	=	Number of telecom elements	
T_MFW_TELECOM *ss_telecom_list	= telecom list		

6.2.31.1 Structure T_MFW_TELECOM

char ss_telecom_type	=		
char ss_telecom_service	=		
char ss_status	=	SS_STAT_ACTIVE	0x01
		SS_STAT_REGISTERED	0x02
		SS_STAT_PROVISIONED	0x04
		SS_STAT_QUIESCENT	0x08
		SS_STAT_UNKNOWN	0xFF

6.2.32 Structure T_MFW_SS_PW_CNF

The Structure is used during call barring for register password operation

enum ss_code	=	SS_CB_BAOC	barr of outgoing calls
		SS_CB_BAOIC	of outgoing international calls
		SS_CB_BAOICexH	of outgoing international calls except HPLMN
		SS_CB_BAIC	of incoming calls
		SS_CB_BAICroam	of incoming calls when roaming
		SS_CB_ALL	all call barring
		SS_CB_ALL_BAOC	all outgoing call barring

	SS_CB_ALL_BAIC	all incoming call barring
enum ss_category	= SS_REGISTRATION	
enum ss_error	= MFW_SS_NO_ERROR	
	MFW_SS_ERROR	
char ss_new_pwd [ ]	= New password	

6.2.33 Structure T_MFW_SS_CLI_CNF

enum ss_code	= SS_CLIP	
	SS_CLIR	
	SS_COLP	
	SS_COLR	
enum ss_category	= SS_INTERROGATION	
enum ss_error	= MFW_SS_NO_ERROR	
	MFW_SS_ERROR	
enum ss_status	= bitmap of SS status:	
	SS_STAT_ACTIVE 0x01	
	SS_STAT_REGISTERED 0x02	
	SS_STAT_PROVISIONED 0x04	
	SS_STAT_QUIESCENT 0x08	
	SS_STAT_UNKNOWN 0xFF	
enum ss_clir_option	= MFW_CL_UNKNOWN	
	MFW_CL_PERM	permanent
	MFW_CL_TEMP_DEF_REST	temporary default restricted
	MFW_CL_TEMP_DEF_ALLOWED	temporary default allowed
enum ss_ovrd_ctgry	= MFW_OV_UNKNOWN	override capacity unknown
	MFW_OV_ENABLED	enabled
	MFW_OV_DISABLED	disabled

6.2.34 Structure T_MFW_SS_USSD

char len	= length of USSD data	
char ussd[ ]	= USSD data	
enum dcs	= MFW_DCS_7bits 0x00	data coding scheme
	MFW_DCS_8bits 0x04	
	MFW_DCS_UCS2 0x08	
	MFW_ASCII	
char error	= error condition	

6.2.35 Structure T_MFW_IMEI

char imei_number[ ]	= long form of IMEI number
char error	= error condition

6.2.36 Structure SatMenu

The Structure is used during SAT menu setup

char nItems	= number of menu items
SatTxt header	= menu header
SatItem items [ ]	= menu items

6.2.37 Structure SatCmdTag

```
char number                 =  command number
char type                   =  command type
char qual                   =  command qualifier

char source                 =  source device
char dest                   =  destination device
union                       =  command data
{
    SatTxt text             =  display text
    SatInput inp            =  input user string
    SatTone tone            =  play tone
    SatMenu menu            =  setup menu
    SatSMS sms              = send SMS
    SatSS ss                =  send SS
    SatCall call            =  setup call
    SatCcRes ccres          =  call control information
    SatData files           =  files to be refreshed
    char fill [448]         =  maximal union size
}
```

6.2.37.1 Structure SatTxt

```
char code                   =  data coding scheme
char len                    =  length of text
int text                    =  offset to text string
```

6.2.37.2 Structure SatAddress

```
char ton                    =  type of number, npi
char len                    =  length of dial number
int number                  =  offset to dial number
```

6.2.37.3 Structure SatData

```
char len                    = length of data
int data                    = offset to data
```

6.2.37.4 Structure SatSmsPdu

```
char len                    = pdu length
int data                    = pdu data
```

6.2.37.5 Structure SatItem

```
char id                     = item identifier
char action                 = next action id
char len                    =  length of text
int text                    =  offset to text data
```

6.2.37.6 Structure SatInput

```
char rspMin                 = minimal response length
char rspMax                 =  maximal response length
SatTxt prompt               = prompt string
SatTxt defRsp               = default response
```

6.2.37.7 Structure SatTone

```
char tone                   = code for tone
char durUnit                = code for duration unit
char durValue               = duration value
SatTxt alpha                = tones alpha tag
```

6.2.37.8 Structure SatMenu

char nItems	= number of menu items
SatTxt header	= menu header
SatItem items [1]	= menu items (min. one)

6.2.37.9 Structure SatSMS

SatTxt info	= user information
SatAddress addr	= address
SatSmsPdu sms	= SMS TPDU

6.2.37.10 Structure SatSS

SatTxt info	= user information
SatAddress ss	= SS string

6.2.37.11 Structure SatCall

SatTxt info	= user information
SatAddress addr	= address
SatData bc	= bearer capabilities
SatData sa	= subaddress
char durUnit	= redial time unit
char durValue	= redial max time

6.2.37.12 Structure SatCcRes

SatAddress addr	= address
SatData bc	= bearer capabilities
SatData sa	= subaddress
SatTxt info	= alpha information
long redialTime	= maximal redial time
int callId	= id of new call
char result	= call control result

6.2.38 Structure T_CGEREP_EVENT_REP_PARAM

T_EVENT_REJECT reject
T_EVENT_ACT act
T_CGCLASS_CLASS mobile_class

6.2.38.1 Structure T_EVENT_REJECT

T_PDP_TYPE pdp_type
T_PDP_ADDRESS pdp_addr

6.2.38.2 Structure T_EVENT_ACT

T_PDP_TYPE pdp_type	
T_PDP_ADDRESS pdp_addr	
CGCLASS_CLASS_OMITTED	= value is omitted
CGCLASS_CLASS_A	= mobile class A (highest)
CGCLASS_CLASS_B	= mobile class B (if necessary consider NET III)
CGCLASS_CLASS_C	= mobile class C (circuit switched)
CGCLASS_CLASS_CG	= mobile class CG (GPRS only mode)
CGCLASS_CLASS_CC	= mobile class CC (circuit switched only mode - lowest)
CGCLASS_CLASS_MAX	= invalid value
SHORT cid	

6.2.39 Structure T_GPRS_CONT_REC

The Structure is used during PDP context setting

T_APN apn	=
T_PDP_TYPE pdp_type	=

T_PDP_ADDRESS pdp_addr =
short d_comp =
short h_comp =
T_QOS qos =
T_QOS min_qos =

6.2.39.1 Structure T_QOS

byte preced
byte delay
byte relclass
byte peak
byte mean

6.2.40 Structure T_MFW_GPRS_CONTEXT

The Structure is used during PDP change of service
USHORT id = context id
T_GPRS_CONT_REC data = context data
BOOL activated = context mode
char *L2P = layer 2 protocol

6.3 Enumerated and Defined Types

The following defines are used as return values when performing Supplementary Services

6.3.1 Check SS string (enum T_MFW_SS_RETURN)

These are the return values when attempting a check of an SS string or attempting a SS via a menu key sequence

MFW_SS_DIAL	start a call
MFW_SS_DIAL_IDX	call via an index
MFW_SS_DTMF	send DTMF message
MFW_SS_USSD	send a un Structured SS descriptor string
MFW_SS_SIM_LOCK	lock or unlock the SIM
MFW_SS_SIM_REG_PW	change PIN1 or PIN2
MFW_SS_REG_PW	register a password
MFW_SS_SIM_UNBLCK_PIN	unblock PIN1 or PIN2
MFW_SS_CF	set call forwarding
MFW_SS_CB	set call barring
MFW_SS_CLIR	set calling line identification restriction
MFW_SS_CLIP	set calling line identification presentation
MFW_SS_COLR	set connected line identification restriction
MFW_SS_COLP	set connected line identification presentation
MFW_SS_WAIT	set call waiting
MFW_SS_MMI	request mobile identifierIMEI
MFW_SS_HOLD	call hold
MFW_SS_MULTIPARTY	call multiparty
MFW_SS_UNKNOWN	unknown string
MFW_SS_FAIL	request cannot be performed

6.3.2 SS thru aCall

These are the return values when attempting a SS via a call

CM_OK	A mobile originated call attempt has been started
CM_EC	A mobile originated emergency call has been started
CM_SIM	The number has started a SIM control procedure
CM_SS	The number has started a supplementary service
CM_USSD	The number has started an un Structured SS
CM_MMI	The number defines a MMI specific procedure
CM_CTRL_STR	control string
CM_CLI	CLI command complete

6.3.3 Registration status (enum T_CGREG_STAT)

These are the parameters associated with the event E_MFW_GPRS_R_REG

CGREG_STAT_NOT_REG	not registered, no searching
CGREG_STAT_REG_HOME	registered, home network
CGREG_STAT_SEARCHING	not registered, but searching
CGREG_STAT_REG_DEN	registration denied
CGREG_STAT_UNKN	unknown
CGREG_STAT_REG_ROAM	registered, roaming