Wireless Internet Platform for Interoperability 2.0.1

- Part 2. HAL API -

September 2004

Korea Wireless Internet Standardization Forum

1. Definition of HAL

In order for the upper layers of the platform to preserve the self-reliance of it's hardware in an abstract level, it calls upon the HAL API to approach the hardware resource. As a consequence, the terminal's native system realizes abstraction, forming a platform within a self-relying hardware. The purpose of HAL is to optimize the transplantation of the O/S, with the HAL API realized on the basis of the native system of each terminal, which makes the transplantation of the entire platform possible without changing the codes of the program within the platform.

For example, a domestic CDMA terminal becomes a terminal platform simply with the porting of HAL over the Qualcomm OS(REX), while Windows becomes an emulator with only the porting of HAL.

Additionally, it is only possible to communicate directly with the hardware only through HAL, which in turn can prevent the system from crashing through a direct access of the hardware using the wrong program.

2. HAL Specification

HAL is classified into several parts according to the subject resource that approaches it, set in an abstract mode for the typical method of operation of each resource. In addition, it also provides the data for WIPI platform's HAL in order to accommodate the native setting of a disparate terminal.

2.1. Variable type

Defining variable type

```
typedef unsigned char       M_Boolean     // unsigned 8bit.
                                          Representing TRUE // FALSE
typedef unsigned int        M_Uint32      // unsigned 32 bit type
typedef unsigned short      M_Uint16      // unsigned 16bit type
typedef unsigned char       M_Uint8       // unsigned 8bit type
typedef signed int          M_Int32       // signed 32bit type
typedef signed short        M_Int16       // signed 16bit type
typedef signed char         M_Int8        // signed 8bit type
typedef signed char         M_Char        // char type
typedef unsigned char       M_Byte        // unsigned 8bit type
typedef signed long long    M_Int64       // signed 64bit type
typedef unsigned long long  M_Uint64      // unsigned 64bit type
typedef unsigned short      M_Ucode       // unsigned 16bit. unicode
                                          // character
typedef signed long         M_Sint32      // signed 32bit type
typedef signed short        M_Sint16      // signed 16bit type
#define NULL          0                   //Defining NULL
#define TRUE          1                   //Defining TRUE
#define FALSE         0                   //Defining FALSE
#define inline __inline                   //Defining inline
```

2.2. API provided by the platform

The following are mathematical functions used on HAL or other tasks, in which the platform realized the relevant API. In most cases, these are needed to transfer the event from other task to the platform task, or to set up a platform. Such is not an API that has to be realized when porting HAL, and must be included in the platform library that is being provided.

<Table 2-2-1> Parameter of the event transfered by HAL

Event	Parameter
MH_KEY_PRESSEVENT MH_KEY_RELEASEEVENT MH_KEY_REPEATEVENT	MH_KeyCode
MH_EXIT_EVENT	NULL
MH_TIMER_EVENT	NULL
MH_SMS_EVENT	Pointer of MH_SMSEvent
MH_ANN_EVENT	Pointer of MH_AnnInfo
MH_CALL_EVENT	Pointer of MH_CallEvent
MH_NETWORK_EVENT	Pointer of MH_NetEvent
MH_SERIAL_EVENT	Pointer of MH_SerialEvent
MH_ MEDIA _EVENT	Pointer of MH_MediaEvent
MH_IODEV_EVENT	Pointer of MH_IODevEvent
MH_GPS_EVENT	Pointer of MH_GPSEvent

MH_KEY_PRESSEVENT

When terminal's button is pressed, the KeyCode value of the button must be altered to a key defined by the MH_KeyCode, and transfered through this function.

MH_KEY_RELEASEEVENT

When terminal's button is released, the KeyCode value of the button must be altered to akey defined by the MH_KeyCode, and transfreed through this function.

MH_KEY_REPEATEVENT

When terminal's button is pressed on for a significant time, the MH_KEYREPEAT_EVENT may be sent to the platform periodically until the releasing the button. If the O/S supports MH_KEYREPEAT_EVENT, the time of the first key repeat and its cycle (time value) must be restored at MH_sysGetInformation ().

MH_EXIT_EVENT

Transfers when the platform's performance is over.

MH_TIMER_EVENT

Transfers when the timer set by MH_timerSet() is expired.

MH_SMS_EVENT

Transfers when new SMS message is received.

MH_ANN_EVENT

Transfers when the data of the terminal's annunciator is renewed.

MH_CALL_EVENT

Transfers a call.

MH_NETWORK_EVENT

Transfers the network event.

MH_SERIAL_EVENT

Transfers the serial event.

MH_ MEDIA _EVENT

Transfers the sound event.

MH_IODEV_EVENT

Transfers the IO device event

MH_GPS_EVENT

Transfers the GPS device event

- **Related data type**

  All events necessary for the platform out of the ones that occur at the O/S are transferred to the platform through the MH_pltEvent() function. enum _MH_Event defines the events to be transferred to the platform. Out of the events defined at enum _MH_Event, detailed events can be transferred with the definition such as enum MH_SUB_XXX_EVENT at each module.

  Ex.) Defining the detail events of MH_SERIAL_EVENT

  enum _MH_SUB_SERIAL_EVENT{

  MH_SERIAL_READ = 0, // When READ interrupt has occured

  MH_SERIAL_WRITE, // event that confirms that write has been enabled at SERIAL

  MH_SERIAL_ERROR, // Serial H/W error

  };

  Refer to the relevant module manual for transfer methods of the defined detail events

  enum _MH_Event {

  MH_EXIT_EVENT = 1, // Event terminating the system

  MH_KEY_PRESSEVENT, // Event informing a key press

  MH_KEY_RELEASEEVENT, // Event informing a key release

  MH_KEY_REPEATEVENT, // Event informing a long key press

  MH_TIMER_EVENT, // Event informing the expiration of the timer

```
    MH_SMS_EVENT,          // Event informing an incoming SMS message
    MH_CALL_EVENT,         // Event informing an incoming phone call
    MH_ANN_EVENT,          // Event informing an alteration of the screen data
    MH_NETWORK_EVENT,      // Event informing the connection and disconnection
                           of the network or socket, including the completion of
                           write/read operation
    MH_SERIAL_EVENT,       // Event informing serial write/read/error status
    MH_MEDIA_EVENT         // Event informing media related buffer or playback
                           status
    };
```

<Table 2-3-1> System related event

Event	Detailed event	Function/situation of event occurance
MH_EXIT_EVENT		Event generated by the O/S to terminate platform
MH_KEY_PRESSEVENT		Key Press
MH_KEY_RELEASEEVENT		Key Release
MH_KEY_REPEATEVENT		Key Press for a certain time
MH_TIMER_EVENT		MH_timerSet()
MH_SMS_EVENT	MH_SMS_NEW	Incoming SMS message
	MH_SMS_SEND_NOTIFY	MH_smsSend()
MH_CALL_EVENT	MH_CALL_INCOMING	Incoming call
	MH_CALL_NOTIFY	MH_callPlace()
MH_ANN_EVENT		Altered status concerning the Annunciator
MH_NETWORK_EVENT	MH_NETEV_NETWORK_OPEN	MH_netConnect()
	MH_ NETEV_NETWORK_CLOSE	MH_netClose()
	MH_ NETEV_SOCKET_CONNECT	MH_netSocketConnect()
	MH_ NETEV_SOCKET_CLOSE	MH_netSocketClose()
	MH_NETEV_SOCKET_READ	MH_netSocketRead()
	MH_NETEV_SOCKET_WRITE	MH_netSocketWrite()
MH_SERIAL_EVENT	MH_SERIAL_READ	MH_serialRead()
	MH_SERIAL_WRITE	MH_serialWrite()

	MH_SERIAL_ERROR	Serial H/W error
	MH_SERIAL_DTR	Altered status of serial cable
MH_MEDIA_EVENT	MH_MDAEV_MEDIA_EMPTY	MH_mdaWriteData() MH_mdaPlay()
	MH_MDAEV_MEDIA_FULL	MH_mdaRecord()
	MH_MDAEV_TONE_EMPTY	MH_mdaTonePlay() MH_mdaFreqTonePlay()
	MH_MDAEV_TONE_ERROR	MH_mdaTonePlay() MH_mdaFreqTonePlay()
MH_IODEV_EVENT	MH_IODEVICEEV_CONNECT MH_IODEVICEEV_TIMEROUT	(*DEVOPENFUN)()
	MH_IODEVICEEV_READ	(*DEVREADFUN)()
	MH_IODEVICEEV_WRITE	(*DEVWRITEFUN)()
	MH_IODEVICEEV_CLOSE	(*DEVCLOSEFUN)()
	MH_IODEVICEEV_ERROR MH_IODEVICEEV_OEMERROR	
MH_GPS_EVENT	MH_GPSEV_SUCCESS	GPS Information Reception Success
	MH_GPSEV_FAILED	GPS Information Reception Failure
	MH_GPSEV_NOTAVAILABLE	GPS Device Not Available
	MH_GPSEV_NOTACKNOWLEDGED	GPS Authenticaltion Failure

```
enum _MH_KeyCode{
MH_KEY_0        = '0',
MH_KEY_1        = '1',
MH_KEY_2        = '2',
MH_KEY_3        = '3',
MH_KEY_4        = '4',
MH_KEY_5        = '5',
MH_KEY_6        = '6',
MH_KEY_7        = '7',
MH_KEY_8        = '8',
MH_KEY_9        = '9',
```

```
MH_KEY_ASTERISK    = '*',
MH_KEY_POUND       = '#',
MH_KEY_UP          = -1,
MH_KEY_DOWN        = -2,
MH_KEY_LEFT        = -3,
MH_KEY_RIGHT       = -4,
MH_KEY_SELECT      = -5,
MH_KEY_SOFT1       = -6,
MH_KEY_SOFT2       = -7,
MH_KEY_SOFT3       = -8,
MH_KEY_SEND        = -10,
MH_KEY_END         = -11,
MH_KEY_POWER       = -12,
MH_KEY_SIDE_UP     = -13,
MH_KEY_SIDE_DOWN= -14,
MH_KEY_SIDE_SEL    = -15,
MH_KEY_CLEAR       = -16,
MH_KEY_FLIPDOWN    = -17,
MH_KEY_FLIPUP      = -18,
MH_KEY_CAMERA      = -19,
MH_KEY_INVALID     = 0
};

enum MH_CallState {
        MH_CS_IDLE = 0,
        MH_CS_CALLING,
        MH_CS_CALLED,
        MH_CS_CALLREJECTED,
        MH_CS_INCOMING,          // Transmitted number to be transferedas
```

```
parameter
        MH_CS_OTHERCALL,        // Standby during phone call
        MH_CS_TRANSFERCALL,     // Transfer to standby call
        MH_CS_END
}

enum  _MH_Annunciator{
MH_ANN_RSSI,            // Outset value specified when current RSSI
                        level is at an update enumeration condition
MH_ANN_BATT,            // Current battery level update
MH_ANN_NOSERVICE,       // Not a serviceable region
MH_ANN_SILENTMODE,      // Silent/ring mode
MH_ANN_ALARM,           // Alarm set condition
}
typedef  _MH_Annunciator  MH_Annunciator;

struct _MH_AnnInfo{
MH_Annunciator type;        //Annunciator info type
        M_Int32 data;
}
typedef  struct _MH_AnnInfo MH_AnnInfo;
```

- **MH_pltEvent**

Prototype

*M_Boolean MH_pltEvent(MH_Event event, void *param)*

Description

This is a function that is used to transfer an event from the other task to the platform. For the specific information of the event that is transferred when using the function is called, refer to the parameter field on the table below:

Parameter

[in]	event	Event transfered to the platform
[in]	param	Detail information value of the applicable event. Refer to the parameter field of the table.

Return Value

Pass

TRUE

Fail

FALSE	Not a compatible event, or when the cue receiving the message is full.

Side Effect

When the event MH_EXIT_EVENT is transfered, the function MH_pltStart() is restored and the platform is terminated.

Reference Item

None

- **MH_pltStart**

Prototype

M_Int32 MH_pltStart(M_Int32 JavaC, M_Char programID, M_Char* path, M_Char* args)*

Description

Function to begin the platform at HAL. When the function is called for, the platform is executed, which is not restored before accepting the MH_EXIT_EVENT event. The parameter designates the very first program to be activated by the platform.

Ex.) When the application manager is programmed with Java, the name of the Main class being “org.kwis.am.Main”, and co-existing on the phone image - MH_pltStart(0, “org.kwis.am.Main”, 0, 0);

Ex.) When the application manager is programmed with Java, the name of the Main class being “org.kwis.am.Main” and existing as an independent file named /test/appManager.jar - MH_pltStart(0, “org.kwis.am.Main”, “/test/appManager.jar”, 0);

Parameters

[in]	JavaC	Java if 0, C program if 1
[in]	programID	Indicating the name of the main class of the program to be executed in case of java, the ID to designate the program to be executed in the case of C
[in]	path	The path value indicating the independent file of the program to be executed. When null, it indicates that the program to be executed exists within the phone image.
[in]	args	Parameter transfered to the program to be executed (0 if there is no parameter transfered)

Return Value

Abnormal termination if negative

Side Effect

None

Reference Item

None

- **MH_pltRegIODevice**

Prototype

*M_Int32 MH_pltRegIODevice(MH_IODevice *dev)*

Description

Registers the Generic I/O device in the platform

Parameters

[in]	dev	Structure where the information on and operation of the device to be registered in the platform are stored

Return Value

Pass

0

Fail

M_E_ERROR	Issued when registration fails

Side Effect

None

Reference Item

MH_IODevice Structure of the Generic I/O HAL API Related Data-Type Description Part

2.3. System

Functions that determine the next operation after being fed the terminal's data or event from the standard platform kernel. Composed of functions that sets the memory domain used by the kernel, or ones that executes the lock mechanism supported by the terminal's O/S, in order to protect the function supporting the debugging console which checks the operating condition of the kernel, as well as the critical section.

- **MH_sysGetHeapBlock**

Prototype

void MH_sysGetHeapBlock (M_Uint32 start, M_Uint32* size)*

Description

Function that determines the volatile memory size and the starting address used by the platform.

The memory must be fixed, and not used by other software modules other than the platform.

Parameters

[out]	start	Restoration of Heap starting address
[out]	size	Restoration of Heap size

Return Value

None

Side Effects

None

Reference Item

None

- **MH_sysGetInformation**

Prototype

M_Int32 MH_sysGetInformation (M_Char command, M_Char* buf, M_Int32 bufSize)*

Description

Function that acquires the system data of the terminal. When the restoring value is a fixed number, it is converted into a decimal string and retored through the buffer. When the manufacturer or telecommunications company wishes to expand the undefined data on the platform, the command value is added in the expansion.

Ex.)M_Char buf[16];

MH_sysGetInformation(”ESN”, buf, sizeof(buf));

Parameters

[in]	command	String value
[out]	buf	Buffer
[out]	bufSize	Buffer size

<Table 2-3-2> System Command

Command	Remark
“ESN”	ESN number
“NID”	Network Identification
“SID”	System Identification
“BASEID”	Base station Identification
“BASELAT”	Base station Latitude
“BASELONG”	Base station longitude
“CURRENTCH”	Current Channel number
“PHONENUMBER”	Phone number
“RSSILEVEL”	Current RSSI level
“BATTERYLEVEL”	Current battery level
“MAXRSSILEVEL”	Maximum RSSI level
“MAXBATTLEVEL”	Maximum battery level
“MAXSERIALNUM”	Maximum number of compatible serial ports
“MAXSOCKETNUM”	Maximum number of compatible sockets

<table>
<tr><td>“MEDIADEVICES”</td><td>Character set for compatible media devices, When varied, divided by “,”. When no compatible devices available, returns M_E_NOTSUP.

Pre-defined character set
<table>
<tr><td>Character set</td><td>Device</td></tr>
<tr><td>“Qualcomm_CMX”</td><td>Qualcomm CMX</td></tr>
<tr><td>“Yamaha_MA1”</td><td>Yamaha MA1</td></tr>
<tr><td>“Yamaha_MA2”</td><td>Yamaha MA2</td></tr>
<tr><td>“Yamaha_MA3”</td><td>Yamaha MA3</td></tr>
<tr><td>“Yamaha_MA5”</td><td>Yamaha MA5</td></tr>
<tr><td>“audio/MIDI”</td><td>If the device is able to playback MIDI format</td></tr>
<tr><td>“audio/MP3”</td><td>If the device is able to playback MP3 format</td></tr>
<tr><td>“IS96”</td><td>QCELP-8K</td></tr>
<tr><td>“IS96A”</td><td>QCELP-8K</td></tr>
<tr><td>“IS733”</td><td>QCELP-13K</td></tr>
<tr><td>“IS127”</td><td>EVRC-8K</td></tr>
</table>
When the device is compatible with the pre-defined character set, the defined charater set is restored, and when incompatible, expanded through the defining process of the vendor or the telecommunications company. If the compatible format is not H/W dependent, it is expanded according to “audio/xxx” and MIME type.

Ex.) If the O/S is compatible with CMX, MA1 EVRC-8K, the restored character set would be ➔ “Qualcomm_CMX, Yamaha_MA1”</td></tr>
<tr><td>“DNS”</td><td>Domain name server designation. IP address character set. Ex.) “127.0.0.1”</td></tr>
<tr><td>“TIMEZONE”</td><td>Restoring current time zone in the formats “GMT+hour:minute “, “GMT-hour:minute“. Hour, minute each used two digit character sets.
Ex.) “GMT+09:30”, “GMT-12:00”</td></tr>
<tr><td>“PHONEMODEL”</td><td>Terminal model ID string
Phone model</td></tr>
<tr><td>“KEYREPEAT”</td><td>“Repeat starting time:Repeat cycle time”, the unit being ‘ms’.
When not compatible, can be restored with M_E_NOTSUP.
Ex.) When “600:250” button is pressed and MH_KEYREPEAT_EVENT first occurs after 600ms, the</td></tr>
</table>

<table>
<tr><td></td><td>event would occur every 250ms until the button is released.</td></tr>
<tr><td>“VIBRATORLEVEL”</td><td>Returns vibrating level compatible with the H/W.
(Minimum 0, Maximum 100)
Ex.) “3” compatible with 3 levels of vibration
“1” compatible with 1 level of vibration</td></tr>
<tr><td>“VOLUMELEVEL”</td><td>Returns volume level compatible with the H/W.
(Minimum 0, Maximum 100)
Ex.) “10” compatible with 10 levels of volume
“4” compatible with 4 levels of volume</td></tr>
<tr><td>“IODEVICES”</td><td>This is the string of the supported I/O device.
In case of several strings, “,” is used to separate them.
When no device is supported, M_E_NOTSUP is returned.
<table>
<tr><td>String</td><td>Device</td></tr>
<tr><td>“IrDA”</td><td>IrDA device</td></tr>
<tr><td>“Camera”</td><td>Camera device</td></tr>
<tr><td>“1ChipCard”</td><td>IC card device for 1Chip</td></tr>
<tr><td>“Bluetooth”</td><td>Bluetooth device</td></tr>
</table>
This returns the defined string when the device supports predefined strings; otherwise, the vendor or mobile communication service provider defines and expands it.</td></tr>
<tr><td>“DEFAULTVOLUME”</td><td>This is the system volume string provided by terminal.
This is the system volume category string provided by terminal.
In case of several strings, “,” is used to separate them.
When no system volume category is supported, M_E_NOTSUP is returned.
<table>
<tr><td>String</td><td>Description</td></tr>
<tr><td>“GENERAL”</td><td>Used for general applications</td></tr>
<tr><td>“VOICE”</td><td>Voice Play/Record feature</td></tr>
<tr><td>“RING”</td><td>This is the ringing feature, e.g., in case the current ringing mode is set to vibration, vibration occurs during playing instead of sounds. In case of a separate melody speaker, sounds occur through it. It is the same as the feature of a terminal receiving a call.</td></tr>
<tr><td>“KEY”</td><td>Key tone feature</td></tr>
<tr><td>“MESSAGE”</td><td>SMS message reception alert feature</td></tr>
<tr><td>“ALARM”</td><td>Alarm alert feature</td></tr>
</table></td></tr>
</table>

	“ALERT”	Various alert features, e.g., no service, low battery, etc.	
	“MMEDIA”	Feature used when playing TCM2, AOD, and VOD	
	“GAME”	Feature used when playing a game	
	“OEM”	Used when setting volume level other than defined above	

Return Value

Pass

0

Fail

M_E_INVALID	Incompatible command value
M_E_NOTSUP	Character to be restored according to the command value does not exist
M_E_SHORTBUF	Storing buffer too short

Side Effects

None

Reference Item

None

- **MH_sysSetInformation**

Prototype

M_Int32 MH_sysSetInformation(M_Char cmd, M_Char* value)*

Description

Function used to alter the system data value. There may be data that cannot be altered according to the terminal's basic software when using HAL.

Parameters

[in]	cmd	Name of the data to be altered
[in]	value	Character set indicating the data value corresponding to cmd

Return Value

Pass

0

Fail

M_E_ACCESS	data that cannot be altered
M_E_INVALID	Wrong cmd or value

Side Effects

None

Reference Item

None

- **MH_sysLock**

Prototype

void MH_sysLock (void)

Description

Function to protect the critical section.

Designates the domain which prohibits the context exterior to the platform. Allows reentry of through the MH_sysUnlock function.

A function such as MH_pltEvent() is called upon for the platform to transfer the event to the platform. When the O/S calls upon MH_pltEvent(), it may do so on the Interrupt Service Routine of the O/S, or even on an other task apart from the platform task. In this case, the critical section between the called MH_pltEvent() function and the platform's context occurs. When MH_sysLock() is called upon, the porting party must call upon HAL in order to prevent the O/S from calling upon MH_pltEvent() before MH_sysUnlock() is done so.

MH_sysLock() and MH_sysUnlock may be called upon in an overlap. In such case, it must be locked when the first MH_sysLock() is called upon, and unlocked when the last MH_sysUnlock is done so.

Parameters

None

Return Value

None

Side Effects

None

Reference Item

MH_sysUnlock

- **MH_sysUnlock**

Prototype

void MH_sysUnlock (void)

Description

Unlocks entry prohibition status at MH_sysLock.

Parameters

None

Return Value

None

Side Effects

None

Reference Item

MH_sysLock

- **MH_sysHalInit**

Prototype

void MH_sysHalInit (void)

Description

Indicates HAL's initiation routine.

This function must be called upon before any HAL API is being used. For API that must be initiated out of those that are compatible to HAL, they are to be done so at this stage.

Parameters

None

Return Value

None

Side Effects

None

Reference Item

None

- **MH_sysHalExit**

Prototype

void MH_sysHalExit()

Description

Function called upon the termination of the platform. Defines all operations to be preceded for each terminal's HAL when the platform terminates, such as releasing the resources used at the HAL level.

Parameters

None

Return Value

None

Side Effects

None

Reference Item

None

- **MH_sysWait**

Prototype

void MH_sysWait (void)

Description

Binary semaphore wait feature

The platform calls upon this function when there is no operation to be processed within itself. When the function is called upon, HAL blocks the task in order to lower the consumption of CPU power.

Parameters

None

Return Value

None

Side Effects

None

Reference Item

None

- **MH_sysSignal**

Prototype

void MH_sysSignal (void)

Description

Binary semaphore signal feature

When this function is called upon, the blocked task at MH_sysWait() must be awakened.

This function is mainly used within MH_pltEvent(). MH_sysSignal() must be ported so that it may operate within the interrupt service routine.

Parameters

None

Return Value

None

Side Effects

None

Reference Item

None

- **MH_debugPutChar**

Prototype

void MH_debugPutChar(M_Char ch);

Description

Outputs the debugging data of the platform. The platform's standard outputs all end up at this API. The HAL operator connects this API to serials, networks and consoles in order to show the message.

Parameters

[in] ch Output character

Return Value

None

Side Effects

None

Reference Item

None

2.4. Timer

The platform generates various timers within the body utilizing a single timer provided by HAL. HAL is able to display from the UTC standard time of January 1st, 1970, 00:00 to the current date and time in milli-second units.

- **MH_timerSet**

Prototype

void MH_timerSet (M_Int64 timeout)

Description

Sets the timer.

When the timer expires, HAL must transfer MH_TIMER_EVENT to the platform. If MH_timerSet() is called upon before the timer expires, the previously set timer is inactivated, while the newly set timer is activated.

Parameters

[in] timeout milli-second unit, 64bit

Return Value

None

Side Effects

None

Reference Item

None

- **MH_timerClear**

Prototype

void MH_timerClear (void)

Description

Inactivates the set timer.

Parameters

None

Return Value

None

Side Effects

None

Reference Item

MH_timerSet

- **MH_timerCurrentTime**

Prototype

M_Int64 MH_timerCurrentTime (void)

Description

Defines the current time.

Parameters

None

Return Value

Time set between UTC standard time of January 1st, 1970, 00:00 to the current date and time, in milli-second units.

Side Effects

None

Reference Item

None

2.5. Unicode

Provides useful character set converting functions on the platform.

- **MH_utilConvertLocalCodeStringToUnicodeString**

Prototype

*M_Int32 MH_utilConvertLocalCodeStringToUnicodeString(M_Uint8 *plocal, M_Int32 len, M_Uint16 *puni, M_Int32 bufSize);*

Description

Converts the local code character set into Unicode character set. When coming across inconvertible character during the conversion process, it changes into a Space(0x20).

Parameters

[in]	plocal	Local “C” character set(character set in local code; EUC_KR for Korea)
[in]	len	Length of the character set
[out]	puni	Buffer on which the converted “Unicode” character set is copied
[in]	bufSize	Length of the buffer

Return Value

Pass

Length of the converted Unicode character set

Fail

M_E_SHORTBUF	When the puni buffer is not sufficient

Side Effects

None

Reference Item

None

- **MH_utilConvertUnicodeStringToLocalCodeString**

Prototype

*M_Int32 MH_utilConvertUnicodeStringToLocalCodeString(M_Uint16 *puni, M_Int32 len, M_Uint8 *plocal, M_Int32 bufSize);*

Description

Converts the Unicode character set to a local code character set. If the Unicode is not compatible with the local code character set, it is converted into 0x20(Space).

Parameters

[in]	puni	“Unicode” character set
[in]	len	Length of the character set
[out]	plocal	Buffer in which the converted “C” character set is copied (character set in local code; EUC_KR for Korea)
[in]	bufSize	Length of the buffer

Return Value

Pass

Length of the converted local code character set

Fail

M_E_SHORTBUF When the plocal buffer is not sufficient

Side Effects

None

Reference Item

None

- **MH_utilUnConvertLocalCodeStringToUnicodeChar**

Prototype

*M_UInt16 MH_utilConvertLocalCodeStringToUnicodeChar(M_Uint8 *plocal, M_Int32 len, M_Int32 *pconsumed);*

Description

Returns the first unicode converted and the local codes used in the first Unicode when converting the local code buffer to Unicode.

Parameters

[in]	plocal	Local code character buffer
[in]	len	Length of the local code character buffer
[out]	pconsumed	Number of local codes used in converting the first unicode

Return Value

Converted Unicode

Side Effects

None

Reference Item

None

- **MH_utilGetLocalCodeSizeInUnicodeString**

Prototype

*M_UInt32 MH_utilGetLocalCodeSizeInUnicodeString(M_Uint16 *puni, M_Int32 len);*

Description

Converts the local code character buffer's size into byte units when converting the Unicode character buffer to a local code character buffer.

Parameters

[in]	puni	Unicode character buffer
[in]	len	Size of the unicode character buffer(Unit : M_Uint16)

Return Value

Size of the local code character buffer needed for conversion (Unit : byte)

Side Effects

None

Reference Item

None

- **MH_utilGetUnicodeSizeInLocalCodeString**

Prototype

*M_UInt32 MH_utilGetUnicodeSizeInLocalCodeString(M_Uint8 *plocal, M_Int32 len);*

Description

Converts the Unicode character buffer size to M_Uint16 unit when converting local code character buffer to unicode character buffer.

Parameters

[in]	psz	Local code character buffer
[in]	len	Length of local code character buffer

Return Value

Size of Unicode buffer needed for conversion (Unit : M_Uint16)

Side Effects

None

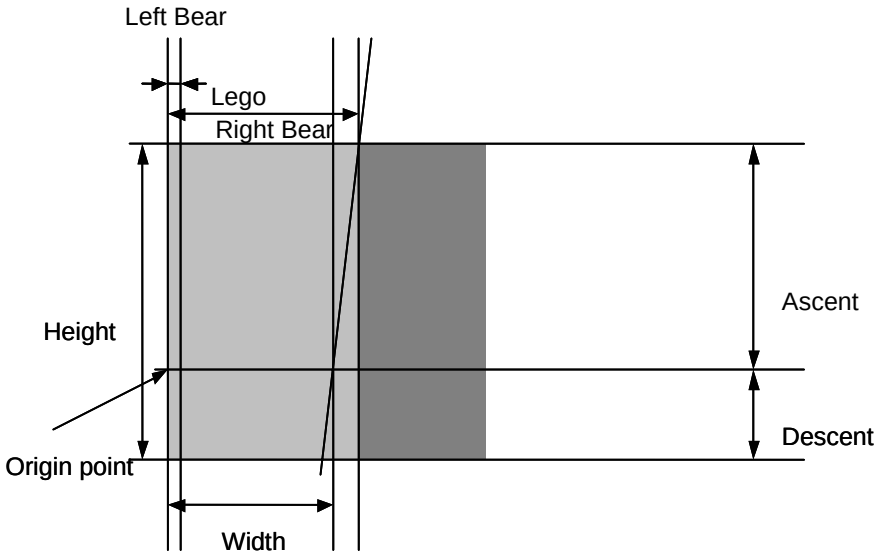
Reference Item

None

2.6. Font

Composed of functions that output the various fonts on screen or obtain various data to do so.

The fonts differ in form, size and face for each system, which is why they are not designated per font, but in abstract. Thus, the actual fonts are obtained according to size(Small|Standard|Large), style(Bold|Itallic|Underlined|Standard), and face, to be conveniently applied onto each terminal.



[Diagram 2-6-1] Font data

Each part of the font has its value added up with it's height, ascent, and descent.

The Bear determines the domain of the image on the actual screen when representing the font. In most cases, the Left Bear is 0, but in the case of the italic font, the Right Bear may be larger than the font's width. The Right Bear is essentially the width of the font's image, with the width deciding the amount of space between the represented font and the next one. If the width is smaller than the Right Bear, there may be an overlap shown in the diagram above.

- **Related data type**

```
typedef struct _MH_CharGlyphInfo {
        int lbear;                  // Left bear of character
        int rbear;                  // Right bear of character
        int width;                  // Width of the character within the screen
        int height;                 // Height of the character within the screen
} MH_CharGlyphInfo;


#define MH_FB_FT_SIZE_SMALL             8
#define MH_FB_FT_SIZE_MEDIUM            0
#define MH_FB_FT_SIZE_LARGE             16
#define MH_FB_FT_FACE_SYSTEM            0
#define MH_FB_FT_FACE_MONOSPACE         32
#define MH_FB_FT_FACE_PROPORTIONAL      64
#define MH_FB_FT_STYLE_PLAIN            0
#define MH_FB_FT_STYLE_BOLD             1
#define MH_FB_FT_STYLE_ITALIC           2
#define MH_FB_FT_STYLE_UNDERLINE        4
```

- **MH_fnGetFont**

Prototype

M_Int32 MH_fnGetFont (M_Int32 face, M_Int32 size, M_Int32 style)

Description

Obtains the ID of the designated font.

The face would be either MH_FB_FT_FACE_SYSTEM, MH_FB_FT_FACE_MONOSPACE(fonts that have fixed widths) or MH_FB_FT_FACE_PROPORTIONAL(fonts that have variable widths), while the size would be either MH_FB_FT_SIZE_SMALL or MH_FB_FT_SIZE_MEDIUM or MH_FB_FT_SIZE_ LARGE. The style would utilize the OR value of MH_FB_FT_STYLE_ITALIC, MH_FB_FT_STYLE_BOLD and MH_FB_FT_STYLE_UNDERLINED, or MH_FB_FT_STYLE_PLAIN. When there isn't a designated font, the closest one is obtained.

Outputing the "C" local character code's entire characters from the designated font ID must be made possible.

Parameters

[in]	face	font face (MH_FB_FT_FACE_SYSTEM, MH_FB_FT_FACE_MONOSPACE, MH_FB_FT_FACE_PROPORTIONAL)
[in]	size	font size (MH_FB_FT_SIZE_SMALL, MH_FB_FT_SIZE_MEDIUM, MH_FB_FT_SIZE_LARGE)
[in]	style	font style(MH_FB_FT_STYLE_ITALIC, MH_FB_FT_STYLE_BOLD, MH_FB_FT_STYLE_UNDERLINED)

Return Value

Font ID

Side Effects

None

Reference Item

None

- **MH_fnGetFontHeight**

Prototype

M_Int32 MH_fnGetFontHeight (M_Int32 font)

Description

Obtains the designated font's height.

Parameters

[in] font Font ID

Return Value

Font height

Side Effects

None

Reference Item

None

- **MH_fnGetFontAscent**

Prototype

M_Int32 MH_fnGetFontAscent (M_Int32 font)

Definition

Obtains the ascent of the designated font

Parameters

[in] font Font ID

Return Value

Ascent of the font

Side Effects

None

Reference Item

None

- **MH_fnGetFontDescent**

Prototype

M_Int32 MH_fnGetFontDescent (M_Int32 font)

Definition

Obtains the descent of the designated font.

Parameters

[in] font Font ID

Return Value

Descent of the font

Side Effects

None

Reference Item

None

- **MH_fnGetCharGlyph**

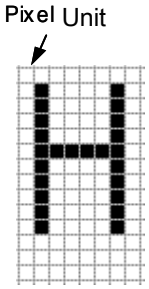
Prototype

*const M_Uint8 *MH_fnGetCharGlyph(const M_Int32 fontid,const M_Uint16 ch,const M_Uint32 fgPxl,const M_Uint32 bgPxl,M_Int32 *pbpl)*

Description

Obtains the image form of the designated character that can be displayed on screen.

The returned buffer has the same number of colors that the main screen LCD has, which therefore is in a form that can be copied straight onto the main screen LCD. The image contents are stored on the buffer that indicates the return value in the order of (0, 0) to (1, 0), (2, 0), ... (0, 1), (1, 1) ... and so forth.



[Diagram 2-6-2] Example of a font

If the LCD supports 16bit color and executes codes such as,

M_Uint8 *pbuf = MH_fnGetCharGlyph(fontid, 'H', blackPixel, whitePixel, &bpl);

M_Uint16 *ppxls = (M_Uint16 *)pbuf;

ppxls[0], ppxls[bpl*1] would be whitePixel, while ppxls[bpl * 1 + 1] would be blackPixel.

Parameters

[in]	fontid	Font ID
[in]	ch	Character (characters are local code, not unicode)
[in]	fgPxl	Foreground colors of the character
[in]	bgPxl	Background colors of the character
[out]	bpl	Number of bytes that are needed per line of the frame buffer

Return Value

Pointer of the image local buffer that is in a form that can be copied. This buffer must not called off on the exterior, nor changed the contents. If there is no character that corresponds to ch, it is returned as NULL.

Side Effects

None

Reference Item

None

- **MH_fnGetCharInfo**

Prototype

*M_Int32 MH_fnGetCharInfo(M_Int32 font, M_Uint16 ch, MH_CharGlyphInfo *pcg)*

Description

Obtains the width, left bear, right bear, and height of the designated character.

Parameters

[in]	font	Font
[in]	char	Character (characters are local code, not unicode)
[out]	pcg	Restoration of character data

Return Value

Pass

0

Fail

If there is no font on the character index that corresponds to M_E_ERROR

Side Effects

None

Reference Item

None

2.7. Character Input Process

Functions related to the character input process. The character (set) is processed according to the current input key value on the automata that has been drawn up to fit each terminal, which is then turned over to the character input process. The character (set) turned over to the character input process function joins the currently arranging charater (set), a completed character (set) would be ready to be transferred.

The character input process function and user text input component manages the insertion, deletion, and modification of the character (set) turned over from the automata.

Information on the input mode that currently supports the automata can be found through MH_IMAgetSurpportModeCount(), MH_IMAgetSupportedModes(), and MH_IMAgetCurrentMode. In the case of MH_IMAgetSupportedModes(), the return value would be the turn over of the language code that is compatible with the automata, with the language code following the ISO 639 code. If the language differentiates capital/small letters, designations of "/S","/L" can be added onto the language code. For example, a small letter in English would be turning over the language code "EN/S".

The Korean language would be turning over the language code "KO". The numbers are not defined in the language code, therefore determined as "N123". As for symbol codes, there are many variable forms of code provided by the phone, hence is not determined at HAL, but realized at an upper level user component.

- **Related data type**

#define MH_IMA_NUM_MODE "N123" // Number input mode

The number input code is designated because numbers are not compatible with the standard language code.

#define MH_IMA_FLUSH –99 // Special key that needs to be force-completed with the characters being arranged by the user

When this key is inputed as MH_IMAhandleInput, the character being arranged is restored in a complete stage. When realizing the automata in the method of having the key inputed and completed a certain period later, this key value may utilize MH_pltEvent to be transferred to the platform.

- **MH_IMAgetSurpportModeCount**

Prototype

M_Int32 MH_IMAgetSurpportModeCount()

Description

Obtains the number of input modes compatible at automata.

Parameters

None

Return Value

Number of input modes

Side Effects

None

Reference Item

None

- **MH_IMAgetSupportedModes()**

Prototype

*M_Char** MH_IMAgetSupportedModes()*

Description

This function obtains the input mode's language code, which is compatible with the automata. The language code follows the ISO 639 code. When a particular language differentiates capital/small letters, "/S" and "/L" can be added to each language code. For example, an English small letter may transfer the "EN/S" language code. For Korean, the "KO" language code is transferred.

Parameters

None

Return Value

Language code (character array pointer)

Side Effects

None

Reference Item

None

- **MH_IMAsetCurrentMode**

Prototype

M_Int32 MH_IMAsetCurrentMode(M_Int32 mode)

Description

This function designates the mode used at the automata. Here, the value is the index value of the language code, which is obtained by MH_IMAgetSupportedModes().

Parameters

Input mode

Return Value

"1" when the designated input mode is correctly applied; otherwise, "0"is returned

Side Effects

None

Reference Item

None

- **MH_IMAgetCurrentMode()**

Prototype

M_Int32 MH_IMAgetCurrentMode(void)

Description

This function obtains the automata's current input mode. Here, the value is the index value of the language code obtained by MH_IMAgetSupportedModes().

Parameters

None

Return Value

Automata's current input mode

Side Effects

None

Reference Item

None

- **MH_IMAhandleInput**

Prototype

*M_Int32 MH_IMAhandleInput(M_Char key, M_Int32 type, M_Char *buf1, M_Int32 *size1, M_Char *buf2, M_Int32 *size2);*

Description

This function processes the key input from the user component according to the current input mode, thereby creating characters and transferring them. If the transferred key cannot be used for the automata's verification, it designates the character in the process of verification to the completed buffer and returns a value of 0.

Note: If MH_IMA_FLUSH is inputted as a key value, the character in the process of verification is competed and returned.

Parameters

[in]	key	Inputted key value (defined in MH_KeyCode, MH_IMA_FLUSH)
[in]	type	Inputted key value (defined in MH_Event)
[out]	buf1	Completed character set buffer
[in]	size1	Size of the completed character set buffer
[out]	buf2	Character set buffer in the process of verification
[in]	size2	Size of the character set buffer in the process of verification

Return Value

TRUE if the automata processes the key; othwerwise, FALSE is returned

Side Effects

None

Reference Item

None

2.8. Virtual Key

These are functions necessary when the terminal's key is used as a virtual functioning key on the application program. For example, if the direction key does not exist on the terminal, these functions are used through mapping. Games and other application programs use keys other than the number keys in operation. Still, whether or not the key other than the number key (control key) exists will vary according to the mobile phone model. Thus, this control key is defined as a virtual key. The virtual functioning key transforms the virtual functioning key value into the actual value and vice versa through the functions MH_keyGetVirtualCode or MH_keyGetKeyCode, enabling the operation of regular application programs without the control key itself. For example, a virtual key coded "MH_VIRGAME_A" may exist corresponding to the "7" key, "1" key, or "SOFT_2" key.

- **Related Data Type**

```
#define MH_VIRUP          1    //  UP functioning key
#define MH_VIRDOWN        6    //  DOWN functioning key
#define MH_VIRLEFT        2    //  LEFT functioning key
#define MH_VIRRIGHT       5    //  RIGHT functioning key
#define MH_VIRFIRE        8    //  FIRE(SEL) functioning key
#define MH_VIRGAME_A      9    //  GAME1 functioning key
#define MH_VIRGAME_B      10   //  GAME2 functioning key
#define MH_VIRGAME_C      11   //  GAME3 functioning key
#define MH_VIRGAME_D      12   //  GAME4 functioning key
#define MH_VIRSIDE_UP     96   //  SIDE UP functioning key
#define MH_VIRSIDE_DOWN   97   //  SIDE DOWN functioning key
#define MH_VIRSIDE_SEL    98   //  SIDE SEL functioning key
#define MH_VIRSIDE_CLEAR  99   //  SIDE CLEAR functioning key
```

- **MH_keyGetVirtualCode**

Prototype

M_Int32 MH_keyGetVirtualCode(M_Int32 keyCode)

Description

Summons the virtual key value mapped onto the actual key value

Parameters

[in]	keyCode	Virtual Keypad

Return Value

KeyCode value of the phone (refer to MH_KeyCode)

Side Effects

None

Reference Item

None

- **MH_keyGetKeyCode**

Prototype

M_Int32 MH_keyGetKeyCode(M_Int32 gameAction)

Description

Summons the actual key value that is mapped onto the virtual key value

Parameters

[in] gameAction Virtual Keypad

Return Value

KeyCode value of the phone (refer to MH_KeyCode)

Side Effects

None

Reference Item

None

2.9. Generic I/O

2.9.1. Summary

- **Definitions**

IrDA

Refers to the device used for infrared ray data transfer

1Chip

Refers to the IC card that saves the personal information necessary for authentication during electronic payment using the terminal

UICC (Universal IC Card)

Refers to the card that saves and manages various terminal data on a WCDMA terminal including the 1Chip feature

- **Summary**

The generic I/O is a standard provision for enabling upgrades of additional I/O devices on the application level without additional API. The functions on each I/O device's basic operations are made up at HAL and registered on the platform. On the application, each device's function registered on the platform is called according to the type of the device for controlling the device itself. The types of devices compatible with the terminal must be restorable, utilizing "IODEVICES" of the MH_sysGetInformation () command.

- **Function Feature and Inventory**

<Table 2-9-1> Function feature and inventory

Categories	Function
Device initialization	MH_ioDevInit()
Device I/O	M_Int32(*DEVOPENFUN)(M_Uint16 devnum, void *param)
	M_Int32(*DEVCLOSEFUN)(M_Uint16 devnum)
	M_Int32(*DEVREADFUN)(M_Uint16 devnum, M_Byte *buf, M_Int32 len)
	M_Int32 (*DEVWRITEFUN)(M_Uint16 devnum, M_Byte *buf, M_Int32 len)
	M_Int32 (*DEVCONTROLFUN)(M_Uint16 devnum, M_Char *cmd, void *param)

- **Device Control Operation**

The basic operations of the device provided include open, close, read, write, and io control. Various operations can also be added upon the registration of the commands on the io control. At HAL, the functions on these operations are drawn up and connected to the MH_IODevice structure and registered on the platform using MH_pltRegIODevice() API.

- **Physical and Logical Device**

The I/O device can be classified into a physical or a logical device depending on its characteristics. The physical device corresponds to the case wherein various devices have the same name. These devices can be physically differentiated, with the application having access to the device using the previously set device number. Physically, the logical device is a single device; nonetheless, it gains access to various ports or channels simultaneously and opens a number of devices in the application without differentiating the device number by the limit that it can possibly accommodate. At HAL, “physical” and “logical” must be stated on the type field to differentiate each one of the devices. At the same time, they must provide the number of devices that can be opened simultaneously. In most cases, a device that can only open one device is considered a physical device. In case of a physical device, the previously set device according to the device number may be opened at HAL. On the other hand, a logical device opens the number of ports and channels that it can possibly accommodate at HAL according to the device number, with HAL administering the mapping table of the device numbers and the opened ports or channels and controlling them accordingly through read/write/control, etc.

- **Compatible I/O Device and Functions of Each Device**

The currently standardized I/O device with high compatibility includes IrDA and 1-chip. The table below shows the available functions and their definitions on OEM of the I/O devices that are currently supporting them. In the case of DEVCONTROLFUN for the I/O device control, the names and features of the commands are explained in detail. The I/O device and command names are fixed values; thus, they cannot be discretely designated by the manufacturer.

<Table 2-9-2> Compatible I/O device and functions for each device

Name of I/O Device	Functions	Definition
“IrDA”	DEVOPENFUN	IrDA device reset
	DEVCLOSEFUN	IrDA device termination
	DEVREADFUN	Data read using IrDA
	DEVWRITEFUN	Data written using IrDA

	DEVCONTROLFUN	“SETOPCODE” – transfer method setup using opcode
“1ChipCard”	DEVOPENFUN	1Chip device reset
	DEVCLOSEFUN	1Chip device termination
	DEVREADFUN	Data read using the 1Chip device
	DEVWRITEFUN	Data written using the 1Chip device
	DEVCONTROLFUN	“GETSTATUS” – Accessibility of the IC card “GETCHANNEL” – Obtaining the logical channel number

2.9.2. Related Data Type

- **MH_IODevEvent**

```
typedef struct MH_IODevEvent {
        M_Int32 event;   // Event that occurs, MH_SUB_IODEVICE_EVENT
        M_Char devname[MH_DEV_NAME_LEN];
                        // Name of the device where an event has occurred
        M_Uint16 devnum;
} MH_IODevEvent;
```

When an event occurs in the system for any of the devices, this data structure is used to transfer the event to the platform.

- **MH_SUB_IODEVICE_EVENT**

```
typedef enum MH_SUB_IODEVICE_EVENT {
        MH_IODEVICEEV_CONNECT = 0,
        MH_IODEVICEEV_READ,
        MH_IODEVICEEV_WRITE,
        MH_IODEVICEEV_CLOSE,
        MH_IODEVICEEV_TIMEOUT,
        MH_IODEVICEEV_ERROR,
        MH_IODEVICEEV_OEMERROR
} MH_SUB_IODEVICE_EVENT;
```

These functions define the type of events that can occur in a particular device.

MH_IODEVICEEV_CONNECT is transferred when a successful connection is made with the party in case of a non-blocking device.

MH_IODEVICEEV_READ and MH_IODEVICEEV_WRITE are transferred at the point where the device can read or write data.

MH_IODEVICEEV_CLOSE is transferred when the connection is terminated.

MH_IODEVICEEV_TIMEOUT is transferred when the given time expires while waiting to be connected with the party.

MH_IODEVICEEV_ERROR is transferred when an abnormal error has occurred.

MH_IODEVICEEV_OEMERROR is transferred when the IO device operation is terminated at OEM under special circumstances such as receiving a phone call.

- **MH_IODevice**

```
typedef struct MH_IODevice {
        M_Char *devname;
        M_Char *devtype;
        M_Uint16 total_devnum;
        DEVOPENFUN open;
        DEVCLOSEFUN close;
        DEVREADFUN read;
        DEVWRITEFUN write;
        DEVCONTROLFUN control;
} MH_IODevice;
```

This function is a data type for registering a single device. A single device is composed of a group of basic operations for controlling the device, name, type, and total number. Whenever a new device is added, this data type is reset and registered on the platform through the MH_pltRegIODevice() function. Physical device types are set as “physical,” and logical device types, as “logical.”

- **DEVOPENFUN**

Prototype

*typedef M_Int32(*DEVOPENFUN)(M_Uint16 devnum, void *param)*

Description

This function opens and rests a particular device. If the device is physically varied, the device is opened through a pre-set devnum. If the devices can open various ports and channels logically, however, they are differentiated through devnum. They discretely allot the port or channel that can be opened at the device, with the mapping table of devnum and port and channel actually allotted and administered separately to enable devnum to control the ports and channel afterwards. A device operating in a non-blocking mode returns M_E_WOULDBLOCK and transfers the MH_IODEVICEEV_CONNECT event to the platform when the device is connected.

Parameters

[in]	devnum	Device number
[in]	param	Parameter passed on when the device is opened

Return Value

Pass

0

Fail

M_E_INVALID	When the parameter is invalid
M_E_WOULDBLOCK	When device connection takes time
M_E_ISCONN	When a particular device is already open
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **DEVCLOSEFUN**

Prototype

*typedef M_Int32(*DEVCLOSEFUN)(M_Uint16 devnum)*

Description

Closes a particular device

Parameters

[in]	devnum	Device number

Return Value

Pass

0

Fail

M_E_ERROR	Error in closing the device

Side Effects

None

Reference Item

None

Left Bear

● DEVREADFUN

Prototype

*typedef M_Int32(*DEVREADFUN)(M_Uint16 devnum, M_Byte *buf, M_Int32 len)*

Description

This function reads data from the device. If a particular device works in a non-blocking mode and fails to read data, M_E_WOULDBLOCK is restored.

Parameters

[in]	devnum	Device number
[out]	buf	Buffer pointer for saving the data read
[in]	len	Size of the buffer

Return Value

Pass

Size of the data read

Fail

M_E_INVALID	When the buffer or its length is invalid
M_E_WOULDBLOCK	When data cannot be read (only when a particular device works in a non-blocking mode)
M_E_ERROR	Other errors

Other error values per device

Side Effects

None

Reference Item

None

- **DEVWRITEFUN**

Prototype

*typedef M_Int32(*DEVWRITEFUN)(M_Uint16 devnum, M_Byte *buf, M_Int32 len)*

Description

This function writes data on the device. If a particular device works in a non-blocking mode and fails to write data, M_E_WOULDBLOCK is restored. As a result, it transfers the MH_IODEVICEEV_WRITE event to the platform at the point where data can be written.

Parameters

[in]	devnum	Device number
[in]	buf	Buffer pointer where data is stored
[in]	len	Size of the buffer

Return Value

Pass

Size of the written data

Fail

M_E_INVALID	When the buffer or its length is invalid
M_E_WOULDBLOCK	When data cannot be written (only when a particular device works in a non-blocking mode)
M_E_ERROR	Other errors

Other error values per device

Side Effects

None

Reference Item

None

- **DEVCONTROLFUN**

Prototype

*typedef M_Int32(*DEVCONTROLFUN)(M_Uint16 devnum, M_Char *cmd, void *param1, void *param2)*

Description

This function controls the device. It is utilized to obtain other information of the device aside from open/close/read/write or to set options.

Parameters

[in]	devnum	Device number
[in]	cmd	Command character set for controlling the device
[in/out]	param1	Parameter for obtaining the value or result for control
[in/out]	param2	Parameter for obtaining the value or result for control

Return Value

Pass

0

Fail

M_E_INVALID	When the parameter is invalid
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_ioDevInit**

Prototype

M_Int32 MH_ioDevInit(void)

Description

This function registers each device through MH_pltRegIODevice(). The open/close/read/write/control functions of each device are enabled to connect to the MH_IODevice structure, which registers the MH_IODevice structure of a particular device utilizing the MH_pltReglODevice() function called within the MH_ioDevInit() function.

Parameters

None

Return Value

Pass

0

Fail

M_E_ERROR Error on reset

Side Effects

None

Reference Item

None

A IrDA Control

- **Device Name**

  “IrDA”

- **Manufacturer’s Terms of Compliance**

  The manufacturer must comply with the IrDA OBEX standard version 1.2 and higher.

  Partition and assembly of OBEX data packet must be done at the terminal. This includes instances wherein the data size corresponding to the OBEX body downloaded from the application is larger than the buffer size that can be processed for the terminal’s OBEX, or when the data buffer size transferred from the terminal to the application is larger than the buffer size that can be read by the application.

- **DEVOPENFUN**

Prototype

*typedef M_Int32(*DEVOPENFUN)(M_Uint16 devnum, void *param)*

Description

This function transfers the MH_IrDAMode structure's pointer to the parameter. An aszMode with a value of "Server" operates in a server mode; defined as "Client," it operates in a client mode. It tries the connection during the period defined by nTime. In case of successful connection, it transfers the MH_IODEVICEEV_CONNECT event to the platform; when a timeout has occurred, however, it transfers the MH_IODEVICEEV_TIMEOUT event.

```
typedef struct MH_IrDAMode {
    M_Char aszMode[MH_IRDA_MODE_LEN];    // Mode designation
    M_Uint32 nTime;                      // Time for connection attempt
} MH_IrDAMode;
```

Parameters

[in]	devnum	Device number
[in]	param	Pointer of the MH_IrDAMode structure

Return Value

Pass

0

Fail

M_E_NOTSUP	When it is incompatible with IrDA
M_E_ISCONN	When a particular device is already connected
M_E_INVALID	When the parameter is invalid
M_E_WOULDBLOCK	When device connection takes time
M_E_ERROR	Other errors

- **DEVCLOSEFUN**

Prototype

*typedef M_Int32(*DEVCLOSEFUN)(M_Uint16 devnum)*

Description

This function closes the IrDA device.

An OBEX Client utilizes OBEX DISCONNECT(0x81) to inform the server of the termination of the connection and consequently terminate the OBEX Client mode session. When the connection is terminated through the party's OBEX connection termination, the event is signaled by transferring the MH_IODEVICEEV_CLOSE event to the platform.

Parameters

Devnum	Device number

Return Value

Pass

0

Fail

M_E_NOTSUP	When it is incompatible with IrDA
M_E_ERROR	Other errors

- **DEVREADFUN**

Prototype

*typedef M_Int32(*DEVREADFUN)(M_Uint16 devnum, M_Byte *buf, M_Int32 len)*

Description

Reads data through the IrDA device

Parameters

[in]	devnum	Device number
[out]	buf	Buffer pointer for saving data (OBEX body) to be read
[in]	len	Size of the buffer

Return Value

Pass

Size of the data read

Fail

M_E_INVALID	When the buffer or its size is invalid
M_E_ERROR	Other errors

Other error values per device

- **DEVWRITEFUN**

Prototype

*typedef M_Int32(*DEVWRITEFUN)(M_Uint16 devnum, M_Byte *buf, M_Int32 len)*

Description

Writes data through the IrDA device

Parameters

[in]	devnum	Device number
[in]	buf	Buffer pointer for saving data (OBEX body) to be written
[in]	len	Size of the buffer

Return Value

Pass

Size of the written data

Fail

M_E_INVALID	When the buffer or its length is invalid
M_E_ERROR	Other errors

Other error values per device

- **DEVCONTROLFUN**

Prototype

*typedef M_Int32(*DEVCONTROLFUN)(M_Uint16 devnum, M_Char *cmd, void *param1, void *param2)*

Description

This function sets the transfer method and uses “SETOPCODE” at cmd.

The buffer pointer allotted on the application is transferred to param1, with the “SETMETHOD” value transferred.

The buffer pointer allotted on the application is transferred to param2, and the value representing a particular pointer will be character set “OBEXPUT” (0x82, OBEX PUT) or “OBEXGET” (0x83, OBEX GET). Other values will be restored as M_E_INVALID.

Unless it is not reset through the control function, the OBEX header value set through “SETOPCODE” maintains a particular opcode value for application to the transfer while the application is being executed. If the write function is called after the IrDA device is open, the OBEX header is constituted by setting the default transfer method to OBEXGET in the absence of a transfer method set by the control function.

Return Value

Pass

0

Fail

M_E_INVALID	When the buffer or its length is invalid
M_E_ERROR	Other errors

B 1-Chip IC Card Control

Device Name

"1ChipCard"

- **DEVOPENFUN**

Prototype

*typedef M_Int32(*DEVOPENFUN)(M_Uint16 devnum, void *param)*

Description

This function activates the 1Chip Card. The pointer of the MH_CardOption structure is transferred to the parameter to obtain the ATR for the 1Chip Card reset. The buffer for obtaining the response is transferred to bATR, whereas the buffer's size is transferred to wATRLen. The actual response value and length are transferred to bATR and wATRLen as a restoration process. If the transferred buffer size is smaller than the actual response value, OxFFFF and M_E_ERROR are restored at wATRLen.

If it is possible to open various logical channels at UICC like the case with WCDMA terminals, channels are allotted according to the device number while the device number and the mapping table of the channel number are administered; thus enabling a particular channel to be controlled by the device number.

```
typedef struct MH_CardOption {
        M_Byte *bATR;          // ATR data pointer
        M_Uint16 wATRLen;      // ATR data length
} MH_CardOption;
```

Return Value

Pass

0

Fail

M_E_INVALID	When the parameter is invalid
M_E_NOCARD	When the card is not inserted
M_E_BADFORMAT	When the transfer data format is invalid
M_E_ERROR	Other errors

- **DEVCLOSEFUN**

Prototype

*typedef M_Int32(*DEVCLOSEFUN)(M_Uint16 devnum)*

Description

Disables the 1Chip Card

Return Value

Pass

0

Fail

M_E_ERROR	Other errors

- **DEVREADFUN**

Prototype

*typedef M_Int32(*DEVREADFUN)(M_Uint16 devnum, M_Byte *buf, M_Int32 len)*

Description

This function reads data through the 1Chip Card. If data cannot be read immediately, the function transfers the MH_IODEVICEEV_READ event to the platform when reading is enabled.

Return Value

Pass

0

Fail

M_E_INVALID	When the parameter is invalid
M_E_NOCARD	When the card is not inserted
M_E_WOULDBLOCK	When data is unreadable
M_E_BADFORMAT	When the transfer data format is invalid
M_E_NOTACTIVE	When the card is not activated
M_E_ERROR	Other errors

- **DEVWRITEFUN**

Prototype

*typedef M_Int32 (*DEVWRITEFUN)(M_Uint16 devnum, M_Byte *buf, M_Int32 len)*

Description

This function writes data through the 1Chip Card. If data cannot be written immediately, the function transfers the MH_IODEVICEEV_WRITE event to the platform when writing is enabled.

Return Value

Pass

0

Fail

M_E_INVALID	When the parameter is invalid
M_E_NOCARD	When the card is not inserted
M_E_WOULDBLOCK	When data cannot be written
M_E_BADFORMAT	When the transfer data format is invalid
M_E_NOTACTIVE	When the card is not activated
M_E_ERROR	Other errors

- **DEVCONTROLFUN**

Prototype

*typedef M_Int32 (*DEVCONTROLFUN)(M_Uint16 devnum, M_Char *cmd, void *param1, void *param2)*

Description

This function checks whether or not the 1Chip Card is inserted

and utilizes “GETSTATUS” at cmd.

The buffer allotted at the application is transferred to param1. A value of “exist” is returned when the card is inserted; otherwise, when the card is not inserted, “noexist” is returned.

The buffer length from param1 is transferred to param2.

Return Value

Pass

0

Fail

M_E_INVALID	When the parameter is invalid
M_E_ERROR	Other errors

Obtaining the UICC’s logical channel number

The function utilizes “GETCHANNEL” at cmd. The fixed number type pointer is transferred to param1. This is used only in the case of a WCDMA terminal, returning the currently allotted UICC logical channel number.

param2 is ignored.

Return Value

Pass

0

Fail

M_E_BADFORMAT	When the transfer data format is invalid
M_E_ERROR	Other errors

2.10. Terminal Resource

2.10.1. Summary of Terminal Resource

- **Definition**

Terminal Resource

Refers to specific data formats such as image, sound, directories, etc., saved at the terminal domain

Terminal Resource Group

Refers to a group of resources classified according to the service providers' own service names (picturemate, musicbell, photo, voice)

Terminal Resource Function

Refers to functions that allow the use of and access to terminal resource group and terminal resource itself

Resource Group Name

Refers to the name constituting the resource group

Resource Name

Refers to the resource name as the sole identifier for differentiating the resources within the resource group; it is created at OEM, and it cannot have the same name within the same resource group.

Resource UI (User Interface) Name

Refers to the name of the resource that appears on the UI screen; it may also be created at the application, allowing duplicate names within the same resource group

Resource Group List

This refers to the list of resource groups, differentiating each resource group's name with the character "\0." Two "\0" characters come in succession at the end of the group name. It is the form used as the output value of the terminal resource function.

Resource List

This refers to the resource name list that differentiates each resource name with "\0." Two "\0" characters come in succession at the end of the group name. It is the form used as the output value of the terminal resource function.

Phonebook/Private

Refers to the resource of the PHONEBOOK/PRIVATE resource group, indicating the terminal's phonebook

Phonebook/Group

Refers to the resource of the PHONEBOOK/GROUP resource group, indicating the terimnal's phonebook group

Short Key

Refers to the resource of the PHONEBOOK/SHORTKEY resource group; it is a simpler number assigned to a particular phone number of a personal phonebook to make phone calling much easier

Group Lock

This refers to the lock status of the resource group. The password must be provided to enable the API to operate when the resource group is group-locked. The group lock can be set/cancelled regardless of the individual lock.

Individual Lock

This refers to the lock status of the resource. The password must be provided to enable the API to operate when the resource is individually locked. The individual lock can be set/cancelled regardless of the group lock.

Lock Setup

This refers to the group or individual lock setup at the resource group or resource. The password must be provided to cancel locks.

Lock Cancellation

This refers to the group lock cancellation at the resource group or the individual lock at the resource. The password must be provided to cancel locks.

Lock Status

Refers to the group lock status for a resource group supporting such feature as well as the individual lock status for a resource supporting such feature

Lock Status and Access to Resource

This refers to the access to the resource such as reading, writing, or deleting data. If the resource group is group-locked, or the resource is individually locked, the password must be provided.

Password

Refers to the password designated to the terminal

Image Resource

Refers to the resources shown on the screen, such as pictures and photos (including continuous image resources such as animated bitmap or soundless video clips)

Sound Resource

Refers to resources that play sounds such as bells, music, etc.

All Resources Including Image and Sound

Refers to all resources such as images shown on the screen or sound that is played back

Specific Status of the Terminal

6 status of the terminal: IDLE, INCOMING, POWERON, POWEROFF, BROWSERON, and BROWSEROFF

Unique ID

Sole ID of the resource allotted by the net or contents provider

Group Data

This refers to a particular resource group data that can be obtained from the terminal resource function. The group data differs according to the resource group and group data type.

Group Data Type

Refers to the type of group data; the group data of the designated resource group can be obtained through the terminal resource function according to the designated group data type

Resource Data

This refers to the particular resource data that can be obtained from the terminal resource function. The resource data varies according to the resource and resource data type.

Resource Data Type

Refers to the type of resource data; it can obtain the resource data of the designated resource through the terminal resource function according to the designated group data type

Query

Refers to the query used in searching the resource through the terminal resource function; when searching for the resource, the terminal resource function returns the resource list with the query character set according to the type of query

Query Type

This refers to the type of query used in searching for resources through the terminal resource function. When searching for the resource, a different type of search is conducted according to the type of search keyword. The resource is then searched using the query type and keyword.

Search Mode

Refers to the search mode for finding the exact match from the given character set or for searching whether the given character set is included in the search result

Write Mode

Refers to the write mode wherein the creation/non-creation of a new resource or overwriting of an existing resource is decided when writing the terminal resource

wCard

Refers to the WIPI phonebook format defined through the expansion of necessary data at the terminal's phonebook, based on vCard 3.0 as the business card data format

A Summary

Data saved at the terminal domain under specific data formats such as image, sound, phonebook, etc., is commonly called the terminal resource (hereinafter referred to as resource).

Resource groups that are classified according to the service provider's own service name (Picturemate, Musicbell, Photo, Voice) are called terminal resource groups (hereinafter referred to as resource group). All resources fall under each resource group.

All resources must be administered as a single storage space domain.

Terminal resource functions provide the access pathway to enable the WIPI application to reach the resource and the resource group.

B Function Features and Index

<Table 2-10-1> Function features and index

Feature	Index
Terminal resource management	MH_termResGetFormat
	MH_termResGetSize
	MH_termResGetUIName
	MH_termResRead
	MH_termResWrite
	MH_termResDelete
	MH_termResRegister
	MH_termResGetRegisteredInfo
	MH_termResGetInfo
	MH_termResSearch
Terminal resource group management	MH_termResGetSupportedGroups
	MH_termResGetCount
	MH_termResGetList
	MH_termResGetGroupInfo
Terminal resource security	MH_termResGetGroupLockState
	MH_termResGetLockState
	MH_termResSetGroupLockState
	MH_termResSetLockState

	MH_termResCheckPassword
Others (terminal resource)	MH_termResGetFreeSpace
	MH_termResExecuteCmd

C Terminal Resource Group

Each resource group is composed of various resources, whereas each resource can have a particular data format. The pre-defined resource groups are as follows:

“PICTUREMATE” – Picturemate

“MUSICBELL” – Musicbell

“PHOTO” – Photo

“VOICE” – Voice recording

“PHONEBOOK/PRIVATE” – Phonebook (private)

“PHONEBOOK/GROUP” – Phonebook (group)

“PHONEBOOK/SHORTKEY” – Short key

“BLACKLIST” – Blacklist; sets the sound/vibration/lamp status of each phone number

“SMSDATA/SENT” – SMS dispatched data

“SMSDATA/RECV” – SMS received data

D List Type

When the terminal resource function is used as values in inputting/outputting various data into a single parameter, each data classification is enabled with the character “\0’(NULL),” with two “\0” characters in succession at the end.

Ex. 1) “PHOTO\0MUSICBELL\0\0”

Ex. 2) “mypicture01\0mypicture02\0...mypictuer100\0\0”

Ex. 3) “MUSICBELL; mymusicbell\0PICTUREMATE;mypicture\0\0”

Ex. 4) “0114441234\001199995555\0Friend\0Family\0\0”

Ex. 5) “1\0100\0\0”

E Terminal Resource Name Type

Access to the terminal resource is made by resource names; OEM must provide a unique resource name as a identifier to be used by the application within the resource group. When creating a new resource, the application cannot designate the resource name; it must only use the resource name provided by OEM.

- **Resource Name Type**

Resource names should be unique. Aside from that, there are no other restrictions.

- **Fixed Resource Name**

The resource name below is fixed, and such given resource names must be used to enable access to a particular resource data at a particular group.

<Table 2-10-2 > Fixed resource name

Resource Group	Resource Name	Remarks
PHONEBOOK/SHORTKEY	Designated shortkey character set	Ex.) “5,” “49”

- **Resource UI (User Interface) Name**

This refers to the name shown on the UI screen. Some resource groups may not have a UI name for the resource depending on the type of resource group (except the PHONEBOOK/SHORTKEY group, which has the same resource name as its UI name).

- **Resource Name and UI Name**

Each resource has a resource name and a UI name. The resource name is the only differentiator for classifying the resource within the resource group. The PHONEBOOK/SHORTKEY group has a fixed standard on the resource name; the terminal administers names to others. The application uses the resource name list provided by the terminal and does the same when creating a new resource. The resource name may be the index or any form of character set, and the terminal only needs to provide a unique name to enable the application to have access to the resource.

On the other hand, the UI name is the name shown to the user on the UI.

For example, in the case of the PHOTO group, the name of the image inputted by the user after taking a picture is the UI name. When the application shows to the user the PHOTO group image list saved in the terminal, the UI name is displayed on the screen. Even when the terminal creates the name of the picture by itself, the name list of each picture must be shown if the picture list needs to be provided by the application. In this case, the UI name is required. Another example is the case of a musicbell from the MUSICBELL group, with resources having UI names such as “Star in My Heart,” “Soyanggang Girl,” and “Come back to Busan Harbor” shown to the user.

Depending on the type of resource, however, there may be instances when a resource does not have a UI name. One example is the SMS DATA. SMS DATA will have a portion of the message displayed on the message list; thus, it does not need to have a UI name.

In other words, the resource name exists as an identifier when using the resource, whereas the UI name is the name shown to the user. While overlapping resource

names in a single resource group are not allowed, it is possible to have two resources with the same UI name. All resources are assigned a resource name, whereas UI names are given only to the resources of a resource group requiring them. Unlike the resource name, the UI name can be set or modified by the application.

F MIME Type and Resource Data Format

Each terminal resource has the following MIME type and data format:

<Table 2-10-3 > Image format

MIME Type	Data Format
image/bmp	Bitmap image format
image/gif	GIF image format
image/jpeg	JPEG image format
image/png	PNG image format
image/sis	SIS image format

<Table 2-10-4 > Animation format

MIME Type	Data Format
anim/sis	SIS image format
anim/gif	GIF image format

<Table 2-10-5 > Video format

MIME Type	Data Format
video/MPEG4	Mpeg4
video/H.263	H.263
video/H.264	H.264

<Table 2-10-6 > Sound format

MIME Type	Data Format
Qualcomm_CMX	Qualcomm CMX
Yamaha_MA1	Yamaha MA1
Yamaha_MA2	Yamaha MA2
Yamaha_MA3	Yamaha MA3 single channel format
Yamaha_MA5	Yamaha MA5
Yamaha_SMAF	Yamaha single channel format
Yamaha_SMAF-phrase	Yamaha multi-channel format

Yamaha_SMAF-audio	Yamaha SMAF-audio format
Audio/MIDI	MIDI
Audio/MP3	MP3
Audio/TONE	Tone
Audio/FREQTONE	Frequency tone
IS96	QCELP-8K
IS96A	QCELP-8K
IS733	QCELP-13K
IS127	EVRC-8K
G.723.1	G.723.1
Audio/AAC	AAC
Audio/AAC+	AAC+
AMR-WB	WCDMA sound codec
AMR-NB	

<Table 2-10-7 > Phonebook (private), phonebook (group), and SHORTKEY format

MIME Type	Data Format
Phonebook/private	wCard character set
Phonebook/group	Actual group name character set: name shown through the UI
Phonebook/shortkey	Shortkey data = <resource name of private phonebook > + “/” + <wCard TEL Type’s Type and Value> Ex.) Given the home phone number of resource name “Kim Chulsoo,” private phonebook resource is 02-1234-5678. “Kim Chulsoo/TEL; TYPE=home: 0212345678”

<Table 2-10-8 > Blacklist format

MIME Type	Data Format
Blacklist	Blacklist data = NUMBER + “;” + STATE NUMBER: reception number (character set form with no “-”) STATE: blacklist setup status “SND”: when set to sound status “VIB”: when set to vibration status “LMP”: when set to lamp status Ex.) When the number 011-1234-5678 is set to lamp status “01112345678; LMP”

<Table 2-10-9 > SMS reception/dispatch data format

MIME Type	Data Format
smsdata	SMS data = INDEX + “\0” + STATE + “\0” + NUMBER + “\0” + DATA + “\0” + TIME + “\0” + “\0” INDEX: SMS message’s internal administration number character set STATE: message status “0”: new message “1”: message already read NUMBER: caller’s or receiver’s phone number character set (with no “-“) DATA: ASCII-type SMS message character set TIME: reception/dispatch time character set (year/month/date/hour/minute (yyyymmddhhmm) format) Ex.) When the message “Hello” is received from the number 011-1234-5678 on 2003 May 4, 12:55 “10\00\001112345678\0Hello\0200305041255\0\0”

G Terminal Resource Storage Space

The function of the terminal resource provided by the platform is to have all resources of the terminal saved to a single storage space. The single terminal resource group should also be able to use the storage space of the restored size through MH_termResGetFreeSpace.

H Terminal Resource Administration

- **Resource Group and Resource**

  All resources must fall under one of the resource groups.

- **Setting the Resource to a Terminal-Specific Status**

  Setting each image resource and sound resource to a terminal-specific status should be enabled.

  Resources with both image and sound must be set as a single resource.

- **PHONEBOOK/SHORTKEY Group's Resource**

  The shortkey resource must exist as a resource, with the phone number set from the shortkeys provided by the terminal. From among usable shortkeys, those that are not in the PHONEBOOK/SHORTKEY group are the ones that are not designated to each phone number. The shortkey resource must be deleted when a phone number in the phonebook (private) or the phonebook (private) itself indicating the shortkey resource is deleted.

- **Shortkey**

  Once the resource list of the PHONEBOOK/SHORTKEY group is obtained, it should be possible to know the shortkey (to which phone numbers are assigned) in use. When infoType of the MH_termResGetGroupInfo function is defined as "RANGE," the shortkey that is compatible with the terminal must be restored. Here, the shortkey that is not in the PHONEBOOK/SHORTKEY group's resource list is a shortkey that is not designated; therefore, it must be provided as one that can be utilized.

- **Phonebook (Private) and Phonebook (Group)**

  When the application writes a phonebook (private), the terminal must designate a particular phonebook group (i.e, "undesignated phonebook group"). . When the application assigns a phonebook (private) to a phonebook group, the phonebook (private) list data pertaining to a particular phonebook group must also be updated.

I Terminal Resource Security Feature

- **Password Check**

  When the application tries to access the resource such as reading, writing, or deleting the resource data of a group-locked resource group or even an individually locked resource group, it must confirm that the password inputted by the user is the same as the one designated on the terminal. In addition, when setting or cancelling the lock feature, the application must check the password first. The terminal provides a password-checking feature using the MH_termResCheckPassword function.

J Functions Applicable to Each Resource Group

The following are the resource groups of API designated by the resource group that can be selected from the terminal resource API (API that does not designate the resource group is not included in the following table):

<Table 2-10-11> Functions applicable to each resource group

Resource Group	API Wherein Designation is Enabled
All resource groups	MH_termResGetCount

	MH_termResGetList MH_termResGetFormat MH_termResGetSize MH_termResRead
Depending on the terminal	MH_termResGetGroupLockState MH_termResSetGroupLockState MH_termResGetLockState MH_termResSetLockState MH_termResGetUIName
Depending on the parameters	MH_termResGetGroupInfo (Depending on infoType) MH_termResGetInfo (depending on infoType) MH_termResSearch (depending on queryType) MH_termResExecuteCmd (depending on the command)
PICTUREMATE MUSICBELL PHOTO VOICE	MH_termResWrite MH_termResDelete MH_termResRegister
PHONEBOOK/PRIVATE PHONEBOOK/SHORTKEY PHONEBOOK/GROUP BLACKLIST SMSDATA/SENT	MH_termResWrite MH_termResDelete
SMSDATA/RECV	MH_termResDelete

- **Lock Status**

  #define MH_TERMRES_GROUP_LOCK

  #define MH_TERMRES_GROUP_UNLOCK

  #define MH_TERMRES_PRIVATE_LOCK

  #define MH_TERMRES_PRIVATE_UNLOCK

  Refers to the lock status of the terminal resource group and terminal resource

- **Search Mode**

  #define MH_TERMRES_EXTSRCH

  #define MH_TERMRES_INCSRCH

  Determines the search mode when searching for the terminal resource

- **Write Mode**

  #define MH_TERMRES_CREATE

  #define MH_TERMRES_UPDATE

  Determines the write mode when writing a terminal resource

- **MH_termResGetSupportedGroups**

Prototype

M_Int32 MH_termResGetSupportedGroups(M_Byte resGroup, M_Int32 bufSize)*

Description

Returns the resource group list that is compatible with the terminal

Parameters

[out]	resGroup	Resource group list (Follow the list format described in the terminal resource overview.)
[in]	bufSize	resGroup buffer size

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_SHORTBUF	When the buffer size matching the return value is short

Side Effects

None

Reference Item

None

- **MH_termResGetCount**

Prototype

M_Int32 MH_termResGetCount(M_Char resGroupName)*

Description

Returns the number of resources pertaining to the designated resource group

Parameters

[in]	resGroupName	Resource group name

Return Value

Pass

Number of resources

Fail

M_E_ERROR	Unknown error
M_E_INVALID	Invalid parameter

Side Effects

None

Reference Item

None

- **MH_termResGetList**

Prototype

M_Int32 MH_termResGetList(M_Char resGroupName, M_Byte* aszList, M_Int32 bufSize)*

Description

Returns the resource list pertaining to the designated resource group

Parameters

[in]	resGroupName	Resource group name
[out]	aszList	Resource name list (Follow the list format described in the terminal resource overview.)
[in]	bufSize	aszList buffer size

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_SHORTBUF	When the transferred buffer size is shorter than the returned character set
M_E_INVALID	Invalid parameter

Side Effects

None

Reference Item

None

- **MH_termResGetFormat**

Prototype

M_Int32 MH_termResGetFormat(M_Char resGroupName, M_Char* resName, M_Char* rtnFormat, M_Int32 rtnFormatSize)*

Description

Returns the MIME-type character set of a designated resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[out]	rtnFormat	MIME-type character set
[in]	rtnFormatSize	rtnFormat buffer size

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_SHORTBUF	When the transferred buffer size is shorter than the returning character set
M_E_INVALID	Invalid parameter

Side Effects

None

Reference Item

None

- **MH_termResGetSize**

Prototype

M_Int32 MH_termResGetSize(M_Char resGroupName, M_Char* resName)*

Description

Returns the data size of the designated resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name

Return Value

Pass

Resource size

Fail

M_E_ERROR	Unknown error
M_E_INVALID	Invalid parameter

Side Effects

None

Reference Item

Utilized to allot the necessary buffer when calling MH_termResRead to read the resource

- **MH_termResGetUIName**

Prototype

M_Int32 MH_termResGetUIName(M_Char resGroupName, M_Char* resName, M_Char* uiName, M_Int32 uiNameSize)*

Description

Returns the UI name of the designated resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[out]	uiName	Name displayed on UI
[in]	uiNameSize	uiName buffer size

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_SHORTBUF	When the transferred buffer size is shorter than the returning character set
M_E_NOTSUP	When the resource group does not support the UI name
M_E_INVALID	Invalid parameter

Side Effects

None

Reference Item

None

- **MH_termResRead**

Prototype

M_Int32 MH_termResRead (M_Char resGroupName, M_Char* resName, M_Byte* pData, M_Int32 bufSize)*

Description

Returns the data of the designated resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[out]	pData	Resource data
[in]	bufSize	pData buffer size

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_SHORTBUF	When the transferred buffer size is shorter than the returning data
M_E_INVALID	Invalid parameter

Side Effects

None

Reference Item

The Caller allots the pData according to the resource size obtained by MH_termResGetSize().

- **MH_termResWrite**

Prototype

M_Int32 MH_termResWrite(M_Char resGroupName, M_Char* resName, M_Int32 nameSize, M_Char* uiName, M_Char* resFormat, M_Byte* pData, M_Int32 bufSize, M_Int32 mode)*

Description

This function writes/updates the resource of the designated resource group. During writing, the function returns the new resource name created; if a UI name is not given, however, the OEM may designate one discretely.

Parameters

[in]	resGroupName	Resource group name
[in/out]	resName	New or existing resource name
[in]	nameSize	resName buffer size
[in]	uiName	Name displayed on UI
[in]	resFormat	Resource's MIME type
[in]	pData	Resource data
[in]	bufSize	pData buffer size
[in]	mode	Write mode

MH_TERMRES_CREATE

Creates a new resource; in this case, resName returns a new resource name

MH_TERMRES_UPDATE

Overwrites an existing resource; in this case, the new resource name is inputted at resName.

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_INSUFSPACE	Insufficient resource storage space
M_E_INVALID	Invalid parameter transfer

M_E_ACCESSDENY	Resource that a user is unauthorized to use
M_E_NOTSUP	Resource group that does not support the write mode
M_E_INVALIDDATA	Data that does not fit a particular data format
M_E_SHORTBUF	When the buffer size is too short to output resName
M_E_MAXCOUNT	When the number of resources reaches the maximum count that the resource group can support

Side Effects

In the MH_TERMRES_CREATE mode, the UI name transferred to the screen may be ignored in case of a particular rule on the UI name at OEM, or if the UI name is not listed in a resource group.

In the MH_TERMRES_UPDATE mode, the function operates when all resource data with existing resource group name, resource name, and MIME type match each other while existing data is updated to new data.

In case of a resource group that is compatible with the UI name, the UI name must be designated. Otherwise, an error (M_E_INVALID) is returned.

Reference Item

The UI name and resource name must be identical at PHONEBOOK/SHORTKEY and PHONEBOOK/GROUP.

- **MH_termResDelete**

Prototype

M_Int32 MH_termResDelete(M_Char resGroupName, M_Char* resName)*

Description

Deletes the designated resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_INVALID	Invalid parameter transfer
M_E_ACCESSDENY	Resource whereby access by user is denied
M_E_NOTSUP	Resource group that does not support the delete feature
M_E_NODELETE	Resource that cannot be deleted

Side Effects

None

Reference Item

None

- **MH_termResRegister**

Prototype

M_Int32 MH_termResRegister(M_Char resGroupName, M_Char* resName, M_Char* szStatus)*

Description

Sets the designated resource to a terminal-specific status

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[in]	szStatus	Terminal-specific status
“IDLE”		Idle mode
“INCOMING”		Incoming call
“POWERON”		Terminal power on
“POWEROFF”		Terminal power off
“BROWSERON”		Browser on
“BROWSEROFF”		Browser off

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_INVALID	Invalid parameter transfer
M_E_ACCESSDENY	User access for setup denied
M_E_INVALIDSTATUS	When the resource and terminal-specific status are not related
M_E_NOTSUP	Resource group that does not support the register feature

Side Effects

None

Reference Item

None

- **MH_termResGetRegisteredInfo**

Prototype

M_Int32 MH_termResGetRegisteredInfo(M_Byte resList,*
M_Int32 bufSize, M_Char szStatus)*

Description

Returns the resource group name and resource name to a terminal- specific status

Parameters

[out]	resList	Resource group and resource name list Composed of a sequence of resource group name, ";," and resource name In case of multiple resources, the following resource group name and resource name are distinguished according to the list form described in the summary of terminal resource. Ex.) "MUSICBELL;mymusicbell\0\0" "MUSICBELL;mymusicbell\0PICTUREMATE;mypicture\0\0"	
[in]	bufSize	resList buffer size	
[in]	szStatus	Terminal-specific status	
		"IDLE"	Idle mode
		"INCOMING"	Incoming call display
		"POWERON"	Terminal power on display
		"POWEROFF"	Terminal power off display
		"BROWSERON"	Browser on
		"BROWSEROFF"	Browser off

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error

M_E_INVALID	Invalid parameter transfer
M_E_ACCESSDENY	User access for setup denied
M_E_INVALIDSTATUS	When the resource and terminal-specific status are not related
M_E_NORES	When there is no designated resource at a specific status
M_E_SHORTBUF	When the resNames buffer size is short
M_E_NOTSUP	Resource group that does not support the register feature

Side Effects

None

Reference Item

Up to two resources may be designated at a terminal-specific status.

- **MH_termResGetGroupLockState**

Prototype

M_Int32 MH_termResGetGroupLockState(M_Char resGroupName)*

Description

Returns the lock status of the designated resource group

Parameters

[in]	resGroupName	Resource group name

Return Value

Pass

MH_TERMRES_GROUP_LOCK	A particular resource group has been group-locked.
MH_TERMRES_GROUP_UNLOCK	A particular resource group has been group-unlocked.

Fail

M_E_ERROR	Unknown error
M_E_NOTSUPPORTLOCK	A particular resource group/ resource does not support the lock mode.
M_E_NOTSUPPORTGLOCK	A particular resource group does not support group lock (supports individual lock).
M_E_INVALID	Invalid parameter transfer

Side Effects

None

Reference Item

None

- **MH_termResGetLockState**

Prototype

M_Int32 MH_termResGetLockState(M_Char resGroupName, M_Char* resName)*

Description

Returns the lock status of a designated resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name

Return Value

Pass

MH_TERMRES_PRIVATE_LOCK	A particular resource has been individually locked.
MH_TERMRES_PRIVATE_UNLOCK	A particular resource has been individually unlocked.

Fail

M_E_ERROR	Unknown error
M_E_NOTSUPPORTLOCK	A particular resource group/ resource does not support the lock mode.
M_E_NOTSUPPORTPLOCK	A particular resource group does not support individual lock (supports group lock).
M_E_INVALID	Invalid parameter transfer

Side Effects

None

Reference Item

None

- **MH_termResSetGroupLockState**

Prototype

M_Int32 MH_termResSetGroupLockState(M_Char resGroupName, M_Int32 state)*

Description

Sets the lock status of a designated resource group

Parameters

[in]	resGroupName	Resource group name
[in]	state	Group lock set/cancellation

MH_TERMRES_GROUP_LOCK	Group lock setup
MH_TERMRES_GROUP_UNLOCK	Group lock cancellation

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_NOTSUPPORTLOCK	A particular resource group/ resource does not support the lock mode.
M_E_NOTSUPPORTGLOCK	A particular resource group does not support group lock (supports individual lock).
M_E_INVALID	Invalid parameter transfer

Side Effects

None

Reference Item

- **MH_termResSetLockState**

Prototype

M_Int32 MH_termResSetLockState(M_Char resGroupName, M_Char* resName, M_Int32 state)*

Description

Sets the lock status of a designated resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[in]	state	Individual lock setup/cancellation

MH_TERMRES_PRIVATE_LOCK	Individual lock setup
MH_TERMRES_PRIVATE_UNLOCK	Individual lock cancellation

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_NOTSUPPORTLOCK	A particular resource group/ resource does not support the lock mode.
M_E_NOTSUPPORTPLOCK	A particular resource group does not support individual lock (supports group lock).
M_E_INVALID	Invalid parameter transfer

Side Effects

None

Reference Item

- **MH_termResCheckPassword**

Prototype

M_Int32 MH_termResCheckPassword(M_Char szPassword)*

Description

Checks whether the designated password matches the terminal's fixed password

Parameters

[in] szPassword Password

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_INCORRECTPASSWORD	Incorrect password
M_E_INVALID	Invalid parameter transfer

Side Effects

None

Reference Item

None

- **MH_termResGetFreeSpace**

Prototype

M_Int32 MH_termResGetFreeSpace(void)

Description

Returns the free space left on the terminal's resource storage space

Parameters

[in] resGroupName Resource group name

Return Value

Pass

Free space left on the designated resource group's storage space

Fail

M_E_INVALID	Invalid parameter transfer
M_E_ERROR	Unknown error
M_E_NOTSUP	Incompatible resource group

Side Effects

None

- **MH_termResGetGroupInfo**

Prototype

M_Int32 MH_termResGetGroupInfo(M_Char resGroupName, M_Char* infoType, M_Byte* infoData, M_Int32 bufSize)*

Description

Returns the group data of the designated group data type for the designated resource group

Parameters

[in]	resGroupName	Resource group name
[in]	infoType	Group data type (see definition at the reference item below)
[out]	infoData	Group data
[in]	bufSize	infoData buffer size

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_INVALID	Invalid parameter transfer
M_E_SHORTBUF	Group data buffer too short
M_E_NOTSUPPORTTYPE	A particular group does not support the group data of the group data type.
M_E_NOENT	Group data type's group data does not exist.

Side Effects

Only the resource group falling under infoType defined at the reference item below can be designated through resGroupName.

Reference Item

"+": refers to the operation working on strcat in front and rear character set/character

<Table 2-10-12>

Resource Group	infoType	Remarks
All groups	“MAXUINAMESIZE”	Returns the maximum storage length of a UI name as a character at the designated resource group
PHONEBOOK/PRIVATE PHONEBOOK/GROUP SMSDATA/SENT SMSDATA/RECV BLACKLIST	“MAXCOUNT”	Returns the maximum storage number of resources at the designated resource group as a character
PHONEBOOK/SHORTKEY	“RANGE”	This returns the first and last shortkey that are compatible with the terminal as a character set. The mode of return is 3.9.1, returning in the order of starting number and ending number. Ex.1) “1\0100\0\0” (terminal that can use any number from 1 to 100 as shortkeys) Ex.2) ”0\0200\0\0” (terminal that can use any number from 1 to 200 as shortkeys)
PHONEBOOK/GROUP	“IRREMOVABLE”	This returns the resource list that is irremovable. The mode of return is a “list form” at the terminal resource module porting. In the absence of irremovable resources, an error (M_E_NOENT) is returned.
‘PHONEBOOK/PRIVATE	“MAXTELCOUNT”	Returns the maximum number of phone numbers that can be stored at the terminal in a character set
	“TYPELIST”	This returns the list of wCard Type that is compatible with the terminal. The mode of return is the “list form” at the terminal resource module porting.
	“TYPECOUNT”	Returns the number of wCard Type that is compatible with the terminal
	“TYPEINFO” + “/” + <Type name> ex) “TYPEINFO/N” “TYPEINFO/TEL”	This returns the data of wCard Type that comes under the Type name of the wCard. The returning mode takes the form of the following character set: <Length of the value that can be stored> + “/” + <Number of a particular Type that can be stored at a single phonebook (private) > [+ “/” + <Type parameter> + “:” <Number of a particular Type parameter that can be stored at a single phonebook (private) >] Examples are as follows:

	"X-MDAYINFO"	Gets anniversary information from the terminal The following is the return format: <Anniversary> + '/' + <Fixed or Variable> (+ '/' + <Maximum length of VARIABLE>) (+ "/NOLEAF") <Anniversary>: YYYYMMDD or MMDD <Fixed or Variable>: FIXED or VARIABLE <Maximum length of Variable>: Maximum length of the anniversary's Type Parameter that can be entered using ASCII CHAR "/NOLEAF": In case of a terminal wherein the leap month cannot be entered in the lunar calendar Refer to wCard Ex. 1) In case the terminal supports YEAR in the anniversary, and the anniversary is FIXED; if the lunar calendar is supported, the leap month is supported as well "YYYYMMDD/FIXED" Ex. 2) In case the terminal supports YEAR in the anniversary, and the anniversary is FIXED; the lunar calendar is supported but not the leap month "YYYYMMDD/FIXED/NOLEAF" Ex. 3) In case the terminal does not support YEAR in the anniversary, and the user directly enters the type of anniversary (maximum available length: 6) "MMDD/VARIABLE/6" The terminal that does not support X-MDAY returns an error (M_E_NOENT).

"TYPEINFO" example

- **Ex. 1) Returning "10/1" as the infoType of "TYPEINFO/N"**

  Based on the standards of the ASCII CHAR, the length of the value that can be stored at "N" Type is 10 words.

  1 "N" Type may be stored at a single phonebook (private).

  No Type parameter exists at "N" Type.

- **Ex. 2) Returning "15/4/cell: 4/work: 4/home: 4" as the infoType for "TYPEINFO/TEL"**

  Based on the standards of the ASCII CHAR, the length of the value that can be stored at "TEL" Type is 15 words.

4 “TEL” Types may be stored at a single phonebook (private).

3 Type parameters (cell/work/home) exist at “TEL” Type.

Type parameters cell, work, and, home may store 4 numbers at a single phonebook (private).

Description: a total of 4 cell/work/home phone numbers may be stored at a single phonebook (private); all 4 may be set as home phone numbers, work phone numbers, or cellular phone numbers.

- **Ex. 3) Returning “10/4/birthday:1/wedding:1/meeting:1/memorial:1/sun:4/ moon:4” as the infoType for “TYPEINFO/X-MDAY”**

  Based on the standards of the ASCII CHAR, the length of the value that can be stored at “X-MDAY” Type is 10 words.

  4 “X-MDAY” Types may be stored at a single phonebook (private).

  6 Type parameters (birthday/wedding/meeting/memorial/sun/moon) exist at “X-MDAY” Type.

  4 Type parameters sun, and moon may store 4 numbers at a single phonebook (private).

  Description: a total of 4 birthdays, wedding anniversaries, meetings, memorial days may be stored at a single phonebook (private). Thus, a total of 4 commemorative dates (birthday, wedding anniversary, meeting, and memorial day) may be stored. All 4 commemorative dates may be stored in either the solar or lunar calendar.

- **MH_termResGetInfo**

Prototype

M_Int32 MH_termResGetInfo(M_Char resGroupName, M_Char* resName, M_Char* infoType, M_Byte* infoData, M_Int32 bufSize)*

Description

Returns the resource data of the designated data type for the designated resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[in]	infoType	Resource data type (see definition at the reference item below)
[out]	infoData	Resource data
[in]	bufSize	infoData buffer size

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_INVALID	Invalid parameter transfer
M_E_SHORTBUF	Resource data buffer size too short
M_E_NOTSUPPORTTYPE	A particular group does not support the group data of the group data type.
M_E_NOENT	Group data type's group data does not exist.

Side Effects

Only the resource falling under the defined resource format at the reference item below can be designated.

Reference Item

"+": refers to the operation working on strcat with the front and rear character set/character

The resource format follows the "resource data format and MIME type" described in the summary of terminal resource.

<Table 2-10-13> Resource format and MIME type

Resource Format	infoType	Remarks
Image format Animation format Video format	“WIDTH”	This returns the width data at the designated resource in character set. The unit is in pixels.
	“HEIGHT”	This returns the height data at the designated resource in character set. The unit is in pixels.
Video format Sound format	“RUNTIME”	This returns the runtime data at the designated resource in character set. The unit is in ms.
	“BITRATE”	This returns the bit rate data at the designated resource in character set. The unit is in bps (bit per sec).
Video format	“FRAMERATE”	This returns the frame rate data at the designated resource in character set. The unit is in fps (frame per sec).
Phonebook/group format	“PRIVATECOUNT”	This returns the number of phonebooks (private) included at the designated phonebook group in character set. A phonebook group without a phonebook (private) returns a value of 0.
	“PRIVATELIST”	This returns the resource name list of the phonebook (private) included at the designated phonebook group in character set. The return format follows the list format described in the summary of terminal resource. A phonebook group without a phonebook (private) returns an error (M_E_NOENT).

- **MH_termResSearch**

Prototype

M_Int32 MH_termResSearch(M_Char resGroupName, M_Char* queryType, M_Char* queryName, M_Byte* resNames, M_Int32 bufSize, M_Int32 mode)*

Description

This function searches the resource matching the given character set query according to resource group and query type. It returns the resource name list as a search result.

Parameters

[in]	resGroupName	Resource group name
[in]	queryType	Query type (see definition at the reference item below)
[in]	queryName	Character set query
[out]	resNames	Resource name list (according to the list form described in the summary of terminal resource)
[in]	bufSize	resNames buffer size
[in]	mode	Search mode

MH_TERMRES_EXTSRCH	Whether it matches the queryName character set
MH_TERMRES_INCSRCH	Whether it includes the queryName character set

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_SHORTBUF	When the transferred buffer is shorter than the returning character set
M_E_INVALID	Invalid parameter transfer
M_E_NOTSUPPORTTYPE	A particular resource group does not support the queryType.
M_E_NOTFOUND	Resource not found

Side Effects

Only the resource group falling under the defined queryType at the reference item below may be designated through resGroupName.

Reference Item

<Table 2-10-14> Resource group and queryType

Resource Group	queryType	Remarks
All resource groups	“UINAME”	Searches the resource of the UI name given the queryName character set from the designated resource group (some resource groups may not support the UI name)
PHONEBOOK/PRIVATE	“NUMBER”	Phone number search: searches the phonebook (private) with the queryName character set from all phone numbers

- **MH_termResExecuteCmd**

Prototype

M_Int32 MH_termResExecuteCmd(M_Char resGroupName, M_Char* cmd, void* param1, void* param2)*

Description

Calls a service according to the designated command

Parameters

[in]	resGroupName	Resource name group that will execute the command
[in]	cmd	Command for calling the service (see definition at the reference item below)
[in/out]	param1	Type/return value of the service (defined on the reference item below)
[in/out]	param2	Type/return value of the service (defined on the reference item below)

Return Value

Pass

0

Fail

M_E_ERROR	Unknown error
M_E_INVALID	Invalid parameter transfer
M_E_NOTSUP	Incompatible command

Side Effects

None

Reference Item

The following are the commands that enable designation; they may be included later.

2.11. Phone Calling

These are functions for making or receiving calls. Any incoming call occurring during the operation of the platform may be received on the platform. The O/S transfers the event to the platform by passing over parameters such as MH_CALL_EVENT and MH_CallEvent with the MH_pltEvent() function.

- **Related Data Type**

```
//Event on phone calling
typedef enum MH_SUB_CALL_EVENT {
MH_CALLEV_INCOMING = 0,   // Incoming call
MH_CALLEV_NOTIFY          // Indicates call disconnection after
                          executing MH_callPlace()
} MH_SUB_CALL_EVENT;

// Structure used to transfer a phone-calling related event
typedef struct MH_CallEvent{
M_Int32 event;        // Event that occurs; enum MH_SUB_CALL_EVENT value
M_Int32 rtnCode;      // 0    MH_callPlace() has been executed successfully;
                      call has been normally disconnected.
                      // -1   MH_callPlace() has not been executed.
}  MH_CallEvent;
```

- **MH_callPlace**

Prototype

*M_Int32 MH_callPlace(M_Char * phonenum)*

Description

This function makes a phone call. Upon successful return, the phone-calling status must be set. After the phone call is disconnected, the termination event (MH_CALLEV_NOTIFY) must be transferred to the platform.

Parameters

[in] phonenum Phone number

Return Value

Pass

0

Fail

M_E_ERROR Phone-calling error

Side Effects

None

Reference Item

None

- **MH_callEnd**

Prototype

void MH_callEnd(void)

Description

This function ends a current phone call or during connection.

Upon termination, the MH_CALLEV_NOTIFY event is transferred to the platform.

Parameters

None

Return Value

None

Side Effects

None

Reference Item

None

- **MH_callAccept**

Prototype

void MH_callAccept(void)

Description

This function receives a requested phone call. If the MH_CALLEV_INCOMING event occurs within the standard platform during the execution of the application program, a phone call may be received using this function.

Parameters

None

Return Value

None

Side Effects

None

Reference Item

MH_SUB_CALL_EVENT

- **MH_callReject**

Prototype

void MH_callReject(void)

Description

This function rejects a requested phone call. If the MH_CALLEV_INCOMING event occurs, a phone call may be rejected using this function.

Parameters

None

Return Value

None

Side Effects

None

Reference Item

None

2.12. Handset Device

These are functions controlling the peripheral devices of the terminal.

- **Related Data Type**

  These are constants related to the control of the backlight.

```
typedef enum MH_DevBackLight {
        MH_LIGHT_ON = 0,            // Backlight on
        MH_LIGHT_OFF,               // Backlight off
        MH_LIGHT_ALWAYS_ON,         // Backlight always on
        MH_LIGHT_DEFAULT            // Backlight set at the user's default mode
} MH_DevBackLight;
```

- **MH_devBacklight**

Prototype

M_Int32 MH_devBacklight (M_Int32 id, MH_DevBackLight on_off, M_Int32 color, M_Int32 timeout)

Description

This function controls the backlight. A preset backlight is assumed to exist for the terminal. After using it on the application, the process is terminated upon restoration to the MH_LIGHT_DEFAULT value to return it to the original state. The backlight goes off at timeout. The system maintains the same status once this function is called; thus, restoring requires the system to use the standard value defined by the user through this function. Backlight number 0 stands for the main LCD backlight, whereas 1 stands for supplementary LCD backlight. If the backlight can designate colors, the colors are designated through the parameter “color,” the form of the value being 0xYYRRGGBB (network byte ordering). Ignoring YY, RR designates red areas, GG, green, and BB, blue.

Parameters

[in]	id	Backlight number
[in]	on_off	Backlight setup option
[in]	color	Backlight color
[in]	timeout	Unit in milliseconds; timeout is only valid given MH_DEV_LIGHT_ON

Return Value

Pass

0

Fail

M_E_ERROR	When the designated backlight does not exist

Side Effects

None

Reference Item

None

- **MH_devVibrator**

Prototype

M_Int32 MH_devVibrator (M_Int32 level, M_Int32 timeout)

Description

This function controls the Vibrator. It switches on the vibrator for a designated time before automatically turning it off.

The timeout value is valid only when the parameter level value is greater than 0. A level value of 0 means that the vibrator is off. The vibration intensity is set by a parameter level value between 0 and 100, with 100 as the most intense vibration supported by the hardware and 0 the least intense. Setting the vibration intensity through a value between 0 and 100 follows the percentage of the vibration level supported by the hardware as shown in the example below. The level of vibration supported by the hardware is returned at MH_sysGetInformation().

Ex.) Hardware with a single vibration level => 1-100: vibration

Hardware with two levels of vibration => 1-50: weak vibration; 51-100: strong vibration

Hardware with three levels of vibration => 1-33: weak vibration; 34-66: medium vibration; 67-100: strong vibration

Parameters

[in]	level	Off when 0, vibration level defined by the O/S when 1-100
[in]	timeout	Vibration time in millisecond

Return Value

Pass

0

Fail

M_E_INUSE	In case the currently requested vibrator is in use
M_E_ERROR	Failure due to other reason

Side Effects

None

Reference Item

None

- **MH_devLedControl**

Prototype

M_Int32 MH_devLedControl(M_Int32 leds, M_Int32 mask)

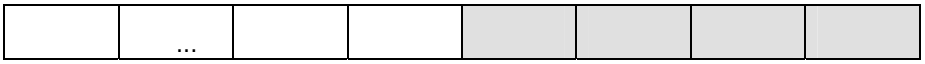
Description

Controls the LED

Ex.)

If there are 4 outer LEDs, this function utilizes 4 beginning with LSB BIT.

31 bit ... 0



With the 4 LEDs of MH_devLedControl(0xF,0x6) as the subject, 2 of these are switched on.

If the terminal supports color screens, LED is set to the brightest value preset by the terminal manufacturer.

Parameters

[in]	leds	LED for control
[in]	mask	Bit to be masked

Return Value

Pass

0

Fail

M_E_INUSE	When the requested LED is in use
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_devGetLed**

Prototype

M_Int32 MH_devGetLed(void)

Description

Reads the LED's on/off status

Parameters

None

Return Value

Returns the bit OR value of each LED's on/off status; if the terminal supports color screens, the on/off value is assigned to each color value by the terminal manufacturer.

Side Effects

None

Reference Item

None

- **MH_devGetLedCount**

Prototype

M_Int32 MH_devGetLedCount(void)

Description

Obtains the outer LED count

Parameters

None

Return Value

LED count

Side Effects

None

Reference Item

None

- **MH_devGetLedSupportColor**

Prototype

M_Int32 MH_devGetLedSupportColor(M_Int32 led, M_Int32 RGB[]

Description

Obtains the color list provided by a particular LED from the terminal

Ex.)

8bit	8bit	8bit	8bit	8bit	8bit	8bit	8bit
RGB[0]				RGB[1]			
Not used	R	G	B	Not used	R	G	B

Parameters

[in]	led	LED for control
[out]	RGB	Buffer with the color value arrangement returned

Return Value

Number of colors provided

Side Effects

None

Reference Item

None

- **MH_devGetLedColor**

Prototype

M_Int32 MH_devGetLedColor(M_Int32 led)

Description

Brings the RGB value designated at a particular current LED

Parameters

[in] led LED for control

Return Value

Preset RGB value

Side Effects

None

Reference Item

None

- **MH_devSetLedColor**

Prototype

M_Int32 MH_devSetLedColor(M_Int32 led, M_Int32 RGB)

Description

This function designates the RGB value from the terminal at a particular LED. In the absence of an RGB value received as a factor, an approximate value is set.

Parameters

[in]	led	LED for control
[in]	RGB	RGB value to be set

Return Value

Pass

0

Fail

M_E_INUSE	When the requested LED is in use
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

2.13. LCD Screen Control

This is composed of functions that print the frame buffer's data onto the LCD screen or obtain the screen's data.

Changing the screen contents in an LCD requires time. Thus, the screen print time is minimized using the double buffering method. MG_fbGetScreenBuffer() enables obtaining the image buffer used in the system. The contents to be printed on the screen is filled in at the acquired memory image buffer, with the contents of the image buffer printed on the LCD screen using MG_fbFlushLcd().

The frame buffer of the terminal's basic software is used.

- **Related Data Type**

typedef struct _MH_DisplayInfo MH_DisplayInfo {

M_Int32 bpp;	// Number of bits in a pixel
M_Int32 depth;	// If the valid number of bits in a pixel is 24, then bpp can be 32, whereas depth can be 24. // A depth of 2 will be the color that can be printed on the actual screen.
M_Int32 width;	// Width of the screen in pixels
M_Int32 height;	// Height of the screen in pixels
M_Int32 bpl;	// Number of bytes in a single line on the screen // Including the PADDING within the interior
M_Int32 colortype;	// Determines whether the LCD is Gray-scale or TRUE COLOR
M_Int32 redmask;	// The mask within the pixel value that will use the red value in case of TRUE color
M_Int32 bluemask;	// The mask within the pixel value that will use the blue value in case of TRUE color
M_Int32 greenmask;	// The mask within the pixel value that will use the green value in case of TRUE color

};

#define MH_FB_MAIN_LCD 1

#define MH_FB_SUB_LCD 2

- **MH_GRP_DIRECT_COLOR_TYPE**

  #define MH_GRP_DIRECT_COLOR_TYPE

  Color type when the palette is not used (defined as 1)

- **MH_GRP_GRAY_TYPE**

  #define MH_GRP_GRAY_TYPE

  Gray-scale type (defined as 2)

- **MH_GRP_COLOR_TYPE**

  #define MH_GRP_COLOR_TYPE

  Color type (defined as 4)

- **MH_fbGetDisplayInfo**

Prototype

void MH_fbGetDisplayInfo(M_Int32 screen,
MH_DisplayInfo displayinfo)*

Description

This function obtains information with reference to the display.

The display information corresponding to the screen is obtained. The screen value may either be MH_FB_MAIN_LCD or MH_FB_SUB_LCD. MH_FB_MAIN_LCD is an LCD that exists in all terminals, whereas MH_FB_SUB_LCD refers to the additional LCD such as the dual folder that may or may not exist depending on the type of terminal.

Parameters

[in]	screen	Screen number; can either be MH_FB_MAIN_LCD or MH_FB_SUB_LCD
[out]	displayinfo	Display information

Return Value

None

Side Effects

None

Reference Item

None

- **MH_fbFlushLcd**

Prototype

void MH_fbFlushLcd(M_Int32 screen, M_Int32 x, M_Int32 y, M_Int32 w, M_Int32 h)

Description

This function prints the contents of the interior screen frame buffer on the LCD.

The contents printed through this function are kept on the screen until the same function is called next time.

Parameters

screen	Screen number; may either be MH_FB_MAIN_LCD or MH_FB_SUB_LCD
x	X coordinates of the area to be printed
y	Y coordinates of the area to be printed
w	Width of the area to be printed
h	Height of the area to be printed

Return Value

None

Side Effects

None

Reference Item

None

- **MH_fbMakePixel**

Prototype

M_Int32 MH_fbMakePixel(M_Int32 color)

Description

Obtains the pixel value corresponding to the designated 0xRRGGBB value

Parameters

color Color value of 0xRRGGBB

Return Value

Pixel value

Side Effects

None

Reference Item

None

- **MH_fbGetPixelFromRGB**

Prototype

M_Int32 MH_fbGetPixelFromRGB(M_Int32 r, M_Int32 g, M_Int32 b)

Description

Obtains the pixel value corresponding to the designated r, g, and b values

Parameters

r	Red (0-255)
g	Green (0-255)
b	Blue (0-255)

Return Value

Pixel value

Side Effects

None

Reference Item

None

- **MH_fbGetRGBFromPixel**

Prototype

*M_Int32 MH_fbGetRGBFromPixel(M_Int32 pixel, M_Int32 *r, M_Int32 *g, M_Int32 *b)*

Description

Obtains the r, g, and b values of the designated pixel value

Parameters

pixel	Pixel value
r	Red (0-255)
g	Green (0-255)
b	Blue (0-255)

Return Value

Pixel value

Side Effects

None

Reference Item

None

- **MH_fbGetScreenBuffer**

Prototype

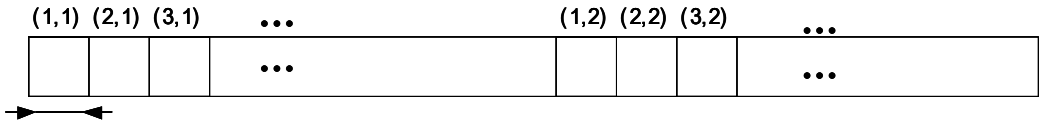
M_Uint8 MH_fbGetScreenBuffer(M_Int32 screen)*

Description

This function returns the screen frame buffer.

The interface rate of the input/output on the LCD is noticeably slow; thus, the contents printed on the LCD must be stored at the host memory. The screen frame buffer used in such system is returned. It should have a size of (bits per pixel * width + padding)/8 * height that allows displaying on LCD and displaying location must not change.

In case of a dual LCD, the main LCD and supplementary LCD usually have varying depth (number of color representation). Nonetheless, the type transferred is assumed to be the same form of data as in the case of the main LCD. If the LCD's data type is different, the contents are printed on the LCD after undergoing a corresponding modification process (printing the closest color on a particular pixel). The data starts on the upper left pixel on the LCD screen, storing the pixel value through the left direction in succession (refer to the diagram below).



Pixel size (2 bytes when LCD is a 16-bit color; 1 byte in case of 8-bit color)

Parameters

[in]	screen	Screen index (MH_FB_MAIN_LCD; main screen, MH_FB_SUB_LCD; supplementary LCD screen)

Return Value

Screen frame buffer

Side Effects

None

Reference Item

None

- **MH_fbSetOEMDisplayArea**

Prototype

M_Int32 MH_fbSetOEMDisplayArea(M_Int32 screenID, M_Int32 x, M_Int32 y, M_Int32 w, M_Int32 h)

Description

This function sets up update protected region on the specific LCD. Once the region is set up,
it is not updated by video media playing device if the media player plays the video media.

Parameters

[in]	screenID	LCD No#, MH_FB_MAIN_LCD, MH_FB_SUB_LCD Either of the two
[in]	x	x axis
[in]	y	y axis
[in]	w	Landscape size
[in]	h	Portrait size

Return Value

Pass

Regional unique RegionID

Fail

M_E_ERROR	Failed due to unknown reason
M_E_INUSE	The specified region is already allocated.
M_E_NOTSUP	Not supported this feature
M_E_INVALID	Incorrect parameter

Side Effects

None

Reference Item

None

- **MH_fbClearOEMDisplayArea**

Prototype

M_Int32 MH_fbClearOEMDisplayArea(M_Int32 regionID)

Description

This function disables update protected region.

Parameters

[in]	regionID	Regional unique ID

Return Value

Pass

0

Fail

M_E_ERROR	Failed due to unknown reason
M_E_NOTEXIST	Incorrect or unregistered ID
M_E_NOTSUP	Not supported this feature in the hardware
M_E_INVALID	Incorrect parameter

Side Effects

None

Reference Item

None

2.14. File

All API file paths approach the unconditional path. All functions related to the file are blocking functions. The file identifier is examined on the platform before calling HAL API; thus, another examination at HAL for an incorrect identifier is not necessary.

- **Related Data Type**

```
#define MH_FILE_OPEN_RDONLY     0x1
#define MH_FILE_OPEN_WRONLY     0x2
#define MH_FILE_OPEN_WRTRUNC    0x4
#define MH_FILE_OPEN_RDWR       0x8
#define MH_FILE_SEEK_SET        0
#define MH_FILE_SEEK_CUR        1
#define MH_FILE_SEEK_END        2
#define MH_FILE_IS_DIR          0x01
#define MH_FILEMODE_RDONLY      0x10
#define MH_FILEMODE_WRONLY      0x20
#define MH_FILEMODE_RDWR        0x30


typedef struct _fileInfo MH_FileInfo {
        M_Int32 attrib          // File attribution bit mask  (directory existence,
                                read-only)
        M_Uint32 creationTime   // File creation time represented in seconds, UTC
        M_Uint32 size           // File size
};
```

- **MH_fileAttribute**

Prototype

M_Int32 MH_fileAttribute(M_Char pathname, MH_FileInfo* fi)*

Description

This function reads the file or directory attribution. For example, this function reads whether the file is a directory or simply a file, what time it was made, etc.

Parameters

[in]	pathname	Unconditional path of the file or directory
[out]	fi	Structure with file attributions

Return Value

Pass

0

Fail

M_E_BADFILENAME	When the path name format is incorrect
M_E_LONGNAME	When the filename exceeds the maximum limit
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_fileAvailable**

Prototype

M_Int32 MH_fileAvailable(void)

Description

Determines the available space of the file system

Parameters

None

Return Value

Pass

Returns the byte size of the available space of the system

Fail

M_E_ERROR Other errors

Side Effects

None

Reference Item

None

- **MH_fileClose**

Prototype

M_Int32 MH_fileClose(M_Int32 fd)

Description

Closes the file

Parameters

[in] fd File identifier

Return Value

Pass

0

Fail

M_E_ERROR

Side Effects

None

Reference Item

None

- **MH_fileList**

Prototype

*M_Int32 MH_fileList(M_Char *root, M_Char* buf, M_Int32 bufSize)*

Description

This function shows the files and lower-level directories within a particular directory. The file and directory names are distinguished by a NULL character (“\0”) in the buffer, ending with two NULL characters in succession.

Parameters

[in]	root	Directory name
[out]	buf	Buffer for filling in the file and directory names
[in]	bufSize	Buffer size

Return Value

Pass

0

Fail

M_E_SHORTBUF	Short buffer size
M_E_BADFILENAME	When the filename form is incorrect
M_E_LONGNAME	When the filename exceeds the maximum length
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_fileMkDir**

Prototype

*M_Int32 MH_fileMkDir(M_Char * dirname)*

Description

Creates the directory

Parameters

[in]	dirname	Unconditional path name of the directory to be created

Return Value

Pass

0

Fail

M_E_BADFILENAME	When the filename form is incorrect
M_E_LONGNAME	When the filename exceeds the maximum length
M_E_NOENT	When an upper-level directory of the directory to be created does not exist
M_E_EXIST	When the directory already exists
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_fileOpen**

Prototype

M_Int32 MH_fileOpen(M_Char pathname, M_Int32 flag)*

Description

This function opens the file. A flag can have the following value:

<Table 2-14-1> File Flag

Flag	Description
MH_FILE_OPEN_RDONLY	Opens the file as “read-only”
MH_FILE_OPEN_WRONLY	Write-only contents are placed at the end of the file.
MH_FILE_OPEN_WRTRUNC	Opens the file as “write-only” and turns the existing file length to 0
MH_FILE_OPEN_RDWR	Opens the file that enables both reading and writing

Parameters

[in]	pathname	File’s unconditional path
[in]	flag	Refer to the table above

Return Value

Pass

Returns the file identifier

Fail

M_E_NOENT	When the file is not found to exist while opening it using MH_FILE_OPEN_RDONLY
M_E_BADFILENAME	When the filename form is incorrect
M_E_LONGNAME	When the filename exceeds the maximum length
M_E_INVALID	When an exceptional option reaches the parameter
M_E_NOSPACE	When there is no available space in the file system
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_fileRead**

Prototype

M_Int32 MH_fileRead(M_Int32 fd, M_Char buf, M_Int32 size)*

Description

Enables reading according to the buffer size of the file

Parameters

[in]	fd	File identifier
[out]	buf	Buffer pointer
[in]	size	Buffer size

Return Value

Pass

Number of bytes read

Returns 0 when size is 0

Returns EOF when case is 0

Fail

M_E_EOF	When read up to the end of the file
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_fileRemove**

Prototype

M_Int32 MH_fileRemove(M_Char pathname)*

Description

This function deletes a file. A file that is in use cannot be deleted, however.

Parameters

[in]	pathname	Unconditional path of the file

Return Value

Pass

0

Fail

M_E_NOENT	When the file does not exist
M_E_INUSE	When the file is in use
M_E_BADFILENAME	When the filename form is incorrect
M_E_LONGNAME	When the filename exceeds the maximum length
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_fileRename**

Prototype

*M_Int32 MH_fileRename(M_Char *oldname, M_Char *newname)*

Description

Changes the filename

Parameters

[in]	oldname	Old unconditional path name
[in]	newname	New unconditional path name

Return Value

Pass

0

Fail

M_E_BADFILENAME	When the filename form is incorrect
M_E_LONGNAME	When the filename exceeds the maximum length
M_E_NOSPACE	When there is no available space in the file system
M_E_ERROR	Other errors
M_E_NOENT	When the file to be changed does not exist
M_E_EXIST	When the recently changed filename already exists
M_E_INUSE	When the filename to be changed is in use

Side Effects

None

Reference Item

None

- **MH_fileRmDir**

Prototype

*M_Int32 MH_fileRmDir(M_Char * dirname)*

Description

Any file or directory must not exist within the directory to be deleted.

Parameters

[in]	dirname	Unconditional path of the directory to be deleted

Return Value

Pass

0

Fail

M_E_BADFILENAME	When the filename form is incorrect
M_E_LONGNAME	When the filename exceeds the maximum length
M_E_NOTEMPTY	When a file or directory exists within the directory
M_E_NOENT	When the directory does not exist
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_fileSeek**

Prototype

M_Int32 MH_fileSeek(M_Int32 fd, M_Int32 pos, M_Int32 where)

Description

This function moves the file pointer to a specific location.

There are 3 methods of calculating the location to be moved: calculating from the very beginning of the file to pos; the very end to pos, and; at the current location to pos.

Parameters

[in]	fd	File identifier
[in]	pos	Location to be moved from the datum point; positive/ negative numbers within the file size are possible
[in]	where	Either one of MH_FILE_SEEK_SET, MH_FILE_SEEK_CUR, or MH_FILE_SEEK_END

Return Value

Pass

Location of the moved file pointer

Fail

M_E_INVALID	When the datum point is not included in either MH_FILE_SEEK_SET, MH_FILE_SEEK_CUR, or MH_FILE_SEEK_END
M_E_BADSEEKPOS	When the file pointer location exceeds the scope of the file
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_fileWrite**

Prototype

M_Int32 MH_fileWrite(M_Int32 fd, M_Char buf, M_Int32 size)*

Description

This function writes the designated size of data from the location indicated by the buffer on the file with identifier fd. When no space is available in the file system for the requested file to be fully written, the used bytes are returned.

Parameters

[in]	fd	File identifier
[in]	buf	Buffer pointer
[in]	size	Byte size to be written

Return Value

Pass

Byte size written

Fail

M_E_ERROR	Other errors
M_E_NOSPACE	When no space is available in the file system

Side Effects

None

Reference Item

None

- **MH_fileTotalSpace**

Prototype

M_Int32 MH_fileTotalSpace(void)

Description

Indicates the size of available space left in the file system

Parameters

None

Return Value

Pass

Returns the byte unit size of the entire space of the file system

Fail

M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_fileSetMode**

Prototype

M_Int32 MH_fileSetMode(M_Char pathName, M_Int32 mode)*

Description

This function changes the attribution of the file. For attribution that can be changed, refer to the fmode value of the parameter table below.

Parameters

[in]	pathName	Unconditional path name of the file
[in]	fmode	File attribution mode

Return Value

Pass

0	Success

Fail

M_E_ERROR	Other errors
M_E_BADFILENAME	When the filename form is incorrect
M_E_LONGNAME	When the filename exceeds the maximum length
M_E_INVALID	Invalid mode
M_E_NOENT	When the file does not exist

Side Effects

None

Reference Item

None

- **MH_fileGetCounts**

Prototype

M_Int32 MH_fileGetCounts(M_Char pathName)*

Description

This function brings the number of files within the directory. The number of files includes the subdirectories.

Parameters

[in]	pathName	Unconditional path name 명 of the directory

Return Value

Pass

Number of files and directories within the directory

Fail

M_E_ACCESS	File inaccessible
M_E_ERROR	Other errors
M_E_BADFILENAME	Bad filename
M_E_LONGNAME	When the directory name exceeds the maximum length

Side Effects

None

Reference Item

None

- **MH_fileIsExist**

Prototype

M_Int32 MH_fileIsExist(M_Char pathName)*

Description

Indicates whether or not a file exists within a specific path

Parameters

[in]	pathName	Unconditional path name of the file

Return Value

Pass

0	File exists.

Fail

M_E_ACCESS	File inaccessible
M_E_ERROR	Other errors
M_E_BADFILENAME	Bad filename
M_E_LONGNAME	When the filename exceeds the maximum length
M_E_NOENT	When the file does not exist

Side Effects

None

Reference Item

None

- **MH_fileTell**

Prototype

M_Int32 MH_fileTell(M_Int32 fd)

Description

Returns the input/output pointer of the current file

Parameters

[in]	fd	File identifier

Return Value

Pass

Current location of the input/output pointer

Fail

M_E_INVALIDFD	Invalid file identifier
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

2.15. Network

This defines the functions supporting TCP/IP Internet telecommunication.

The platform supports a unitary network interface for TCP/IP Internet telecommunication. The platform must enable data communication through the Internet via the socket-related API after calling the MH_netConnect() function to provide Internet connection. After calling MH_netClose() from the platform, data communication through any socket API must be non-executable. It is recommended that all sockets using the network be terminated when MH_netClose() is called on the platform.

Owing to the characteristics of the network, the platform may freeze if the network API is called and blocked for a significant period of time. Thus, the network API must be realized through non-blocking functions except MH_netClose() and MH_netSocketClose(). MH_netClose() and MH_netSocketClose() blocking mean that the related event is not transferred to the platform after this function is called.

When the network API is called, and an I/O event occurs, event-related data must be transferred to the platform. Here, the MH_pltEvent() function is called, and MH_NETWORK_EVENT and MH_NetEvent type data are transferred through the parameter.

- **Related Data Type**

```
#define MH_AF_INET    2          // Constant indicating the Internet domain


// Constant differentiating the TCP/UDP socket

typedef enum MH_SOCKET_TYPE{
        MH_SOCKET_STREAM =1,      //  TCP SOCKET
        MH_SOCKET_DGRAM           //  UDP SOCKET

} MH_SOCKET_TYPE;


// Network event

typedef enum MH_SUB_NETWORK_EVENT {

MH_NETEV_NETWORK_OPEN = 0x01,          // Network connection event

MH_NETEV_NETWORK_CLOSE = 0x02,         // Network termination event
MH_NETEV_SOCKET_CONNECT = 0x04,        // SOCKET connection
eventMH_NETEV_SOCKET_CLOSE = 0x08,     // SOCKET termination
```

```
eventMH_NETEV_SOCKET_READ = 0x10,       // SOCKET READ
eventMH_NETEV_SOCKET_WRITE = 0x20,      // SOCKET WRITE event
} MH_SUB_NETWORK_EVENT;

// Structure transferring the network event
typedef struct MH_NetEvent {
M_Int32 fd;        // Socket identifier
M_Int32 event;  // Event that occurs, MH_SUB_NETWORK_EVENT value
} MH_NetEvent;
```

- **MH_netConnect**

Prototype

M_Int32 MH_netConnect(void)

Description

This function attempts to access the Internet. A return value of M_E_WOULDBLOCK means that Internet access is enabled by non-blocking. When Internet access is enabled, the MH_NETEV_NETWORK_OPEN event must be transferred to the platform. If for any reason Internet access is prohibited, the MH_NETEV_NETWORK_CLOSE event must be transferred to the platform.

Parameters

None

Return Value

Pass

0

Fail

M_E_WOULDBLOCK	When Internet access takes some time
M_E_INPROGRESS	When attempting to gain Internet access
M_E_ISCONN	When Internet access has already been enabled
M_E_ERROR	When Internet access is terminated, or an error has occurred for any other reason

Side Effects

None

Reference Item

None

- **MH_netClose**

Prototype

M_Int32 MH_netClose(void)

Description

This blocking function terminates the Internet access. Even when Internet access is prohibited, success is returned. After the platform calls MH_netClose(), the O/S should prohibit data communication using the Internet through any socket API. It is recommended that all sockets being used in networks be terminated when MH_netClose() is called.

Parameters

None

Return Value

Pass

0

Fail

M_E_INPROGRESS	When terminating Internet access
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_netSocket**

Prototype

M_Int32 MH_netSocket(M_Int32 domain, M_Int32 type)

Description

This function obtains the socket. The socket identifier returned by this function cannot be a negative number.

Parameters

[in]	domain	MH_AF_INET in case of Internet domain
[in]	type	MH_SOCKET_STREAM (TCP socket), MH_SOCKET_DGRAM (UDP socket)

Return Value

Pass

Socket identifier

Fail

M_E_NOTSUP	In case of unsupported domain or type
M_E_NOTCONN	When Internet access is prohibited
M_E_NOSPACE	When the maximum amount of socket allotments has been made
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_netSocketConnect**

Prototype

M_Int32 MH_netSocketConnect(M_Int32 fd, M_Int32 addr, M_Uint16 port)

Description

This function connects the TCP socket to the server. A return value of M_E_WOULDBLOCK means that the socket is connected in a non-blocking mode; when the socket is connected, the MH_NETEV_SOCKET_CONNECT event must be transferred to the platform.

Parameters

[in]	fd	Socket identifier
[in]	addr	IP address of the party; network byte order
[in]	port	Port number of the party; network byte order

Return Value

Pass

0

Fail

M_E_INVALID	When the address is 0
M_E_WOULDBLOCK	Queued for socket to be set connected
M_E_NOTCONN	When Internet access is prohibited
M_E_ISCONN	When connection to the socket already exists
M_E_INPROGRESS	When connection to the server is in progress
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_netSocketRead**

Prototype

*M_Int32 MH_netSocketRead(M_Int32 fd, M_Byte *buf, M_Int32 len)*

Description

This function reads data through TCP. A Return Value of 0 means the end of the file. A Return Value of M_E_WOULDBLOCK means that there is no data to be read. In this case, the MH_NETEV_SOCKET_READ event must be transferred to the platform at the point where data can be read.

Parameters

[in]	fd	Socket identifier
[out]	buf	Data's buffer pointer
[in]	len	Size of the buffer to be read

Return Value

Pass

Length of the read data

Fail

M_E_WOULDBLOCK	When there is no data to be read
M_E_NOTCONN	When the socket is not connected, or Internet access is prohibited
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_netSocketWrite**

Prototype

*M_Int32 MH_netSocketWrite(M_Int32 fd, M_Byte *buf, M_Int32 len)*

Description

This function sends data through the TCP socket. A Return Value of M_E_WOULDBLOCK means that the system's interior causes the temporary failure of data to be transferred. In this case, the MH_NETEV_SOCKET_WRITE event must be transferred to the platform at the point where data can be transferred.

Parameters

[in]	fd	Socket identifier
[in]	buf	Data's buffer pointer
[in]	len	Data size

Return Value

Pass

Length transferred

Fail

M_E_WOULDBLOCK	When data cannot be transferred temporarily due to the system's interior
M_E_NOTCONN	When the socket is not connected, or Internet access is prohibited
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_netSocketBind**

Prototype

M_Int32 MH_netSocketBind(M_Int32 fd, M_Uint32 addr, M_Uint16 port)

Description

Designates the local port at the socket

Parameters

[in]	fd	Socket identifier
[in]	addr	IP address of the party; network byte order
[in]	port	Local port number; network byte order

Return Value

Pass

0

Fail

M_E_INVALID	When the address or port is invalid
M_E_ISCONN	When the port is already designated
M_E_NOTCONN	When Internet access is prohibited
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_netSocketSendTo**

Prototype

*M_Int32 MH_netSocketSendTo(M_Int32 fd, M_Byte *buf, M_Int32 len, M_Int32 addr, M_Uint16 port,)*

Description

This function transfers data through the UDP socket. A Return Value of M_E_WOULDBLOCK means that the system's interior causes the temporary failure of data to be transferred. In this case, the MH_NETEV_SOCKET_WRITE event must be transferred to the platform at the point where data can be transferred.

Parameters

[in]	fd	Socket identifier value
[in]	buf	Data's buffer pointer
[in]	len	Data size
[in]	addr	IP address of the party; network byte order
[in]	port	Port number of the party; network byte order

Return Value

Pass

Transferred length of data

Fail

M_E_WOULDBLOCK	When transfer takes time
M_E_NOTCONN	When Internet access is prohibited
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_netSocketRcvFrom**

Prototype

*M_Int32 MH_netSocketRcvFrom(M_Int32 fd, M_Byte *buf, M_Int32 len, M_Int32 *addr, M_Uint16 *port)*

Description

This function receives data through the UDP socket. A Return Value of M_E_WOULDBLOCK means that there is no data to be read. In this case, the MH_NETEV_SOCKET_READ event must be transferred to the platform at the point where data can be transferred.

Parameters

[in]	fd	Socket identifier value
[in]	buf	Data's buffer pointer
[in]	len	Data size
[in]	addr	IP address of the party; network byte order
[in]	port	Port number of the party; network byte order

Return Value

Pass

Data length received

Fail

M_E_WOULDBLOCK	When there is no data to be read
M_E_NOTCONN	When Internet access is prohibited
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_netSocketClose**

Prototype

M_Int32 MH_netSocketClose(M_Int32 fd)

Description

This blocking function terminates the socket.

Parameters

[in]	fd	Socket identifier value

Return Value

Pass

0

Fail

M_E_INPROGRESS	When termination is in progress
M_E_NOTCONN	When Internet access is prohibited
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_netGetMaxPacketLength**

Prototype

M_Int32 MH_netGetMaxPacketLength(void)

Description

Obtains the maximum length of UDP packet that can be transmitted/received

Parameters

None

Return Value

Packet length

Side Effects

None

Reference Item

None

- **MH_netSocketAccept**

Prototype

M_Int32 MH_netSocketAccept(M_Int32 fd)

Description

This function returns the socket connected with the client through the TCP server socket. Before this function is called, the local port must be designated first through MH_netSocketBind(). If this function returns M_E_WOULDBLOCK, and when a socket connected to the client is created later, the O/S must transfer the MH_NETEV_SOCKET_CONNECT event to the platform. This function's support is an option for platform realization.

Parameters

fd – TCP Server socket identifier

Return Value

Pass

Socket identifier connected with the client

Fail

M_E_WOULDBLOCK	When connection with the client takes time
M_E_NOTSUP	When the platform does not support this function, or the function does not support the socket
M_E_NOTCONN	When Internet access is prohibited
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

2.16. Serial Communication

These are functions that control the serial port supported by the terminal.

The number of serial ports compatible with the terminal may be determined through the MH_sysGetInformation() function. The serial port numbers are given through the method “First serial port = 0, second serial port = 1, etc. (maximum compatible port - 1).” If the status of the serial port is altered, the O/S transfers the event to the platform to report such alteration. The MH_pltEvent() function is called by the O/S to transfer the event to the platform. In this case, it must pass on MH_SERIAL_EVENT and MH_SerialEvent as parameters.

- **Related Data Type**

```
//Serial event
typedef enum MH_SUB_SERIAL_EVENT {
        MH_SERIALEV_READ = 0,     // Event transferred when reception is
                                  enabled
        MH_SERIALEV_WRITE,        // Event transferred when transmission is
                                  enabled
        MH_SERIALEV_ERROR,        // Serial H/W error
} MH_SUB_SERIAL_EVENT;

//Structure transferring the serial event
typedef struct MH_SerialEvent {
M_Int32 fd;             // Port identifier
M_Int32 event;          // Event that occurs, MH_SUB_SERIAL_EVENT value
} MH_SerialEvent;
```

- **MH_serialOpen**

Prototype

*M_Int32 MH_serialOpen (M_Int32 port, M_Char *param)*

Description

This function enables the serial to open the port. The designated port is controlled by the control character set transferred to the parameter. The control character set must have the following format:
Control character set::= [control pair]
Control pair::= Control key “=” control value *(control key “=” control value)
Control key::= Character set made up of alphabets or numbers
Control value::= Character set made up of alphabets or numbers

The control key and value defined in this document are shown in the table below.

<Table 2-16-1> Serial control key and control value

Control Key	Control Value	Description	Default
Baudrate	9600, 19200, 38400, 57600, 115200	Baud rate	115200
Parity	Even, odd, no	Parity	No
Size	Number	Byte size	8
Flow	Hardware, software, no	Flow control	No

Definitions on other control keys and values may be added. For example, opening the port with conditions 8 bit, paritynone, and 115200 speed hardware flow control may be composed of the following control character set:
”baudrate=115200, parity=no, size=8, flow=hardware”
If the defined control key does not exist in the control character set, it is set as a default value.

Parameters

[in] port Port number

[in] param Control character set

Return Value

Pass

Returns the identifier

Fail

M_E_NOTSUP	When control for a particular port number of the control character set is not provided
M_E_INVALID	When the control character set has an invalid format
M_E_ISCONN	When a particular port is already open
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_serialWrite**

Prototype

M_Int32 MH_serialWrite(M_Int32 fd, M_Uint8 buf, M_Int32 len)*

Description

This function transfers data to the serial port.

A Return Value of M_E_WOULDBLOCK means that the transfer cannot be made temporarily due to the system interior. When transfer is enabled, the MH_SERIALEV_WRITE event must be transferred to the platform. Likewise, when the platform receives the MH_SERIALEV_WRITE event, it may retry writing the serial data by calling MH_serialWrite() again. If this function is called, and the party is not perceived at the DTR pin, M_E_NOTCONN is returned.

Parameters

[in]	fd	Serial identifier
[in]	buf	Data buffer
[in]	len	Size of transfer

Return Value

Pass

Transferred size

Fail

M_E_ERROR	Other errors
M_E_WOULDBLOCK	When transfer takes time
M_E_NOTCONN	When the party is not perceived at the DTR pin after calling this function

Side Effects

None

Reference Item

None

- **MH_serialRead**

Prototype

*M_Int32 MH_serialRead (M_Int32 fd, M_Uint8 *buf, M_Int32 len)*

Description

This function receives data through the serial port. A Return Value of M_E_WOULDBLOCK means that there is no data to be read. When data arrives, the MH_SERIALEV_READ event must be transferred to the platform. If the platform receives the MH_SERIALEV_READ event, it may retry reading the serial data by calling MH_serialRead() again. M_E_NOTCONN is returned if this function is called, and the party is not perceived at the DTR pin.

Parameters

[in]	port	Serial identifier
[out]	buf	Data buffer
[in]	len	Buffer length

Return Value

Pass

Length of the data read

Fail

M_E_ERROR	Other errors
M_E_WOULDBLOCK	When the arrival of readable data takes time
M_E_NOTCONN	When the party is not perceived at the DTR pin after calling this function

Side Effects

None

Reference Item

None

- **MH_serialClose**

Prototype

M_Int32 MH_serialClose(M_Int32 fd)

Description

Closes the serial port

Parameters

[in]	fd	Serial identifier

Return Value

Pass

0

Fail

M_E_BADFD	When the serial identifier is invalid
M_E_ERROR	When an error has occurred in closing the serial port

Reference Item

None

2.17. Media Handler

These are APIs for supporting the media handler processing all media data including sound and motion pictures. Media handlers refer to devices that produce/consume data as strip. Sound devices, vocoder devices, camera devices, etc., are part of these devices.

- **Related Data Type**

```
// Tone value
typedef enum MH_mdaToneType {
    MH_SND_TONE_0 = 0,          // DTMF for 0 key
    MH_SND_TONE_1,              // DTMF for 1 key
    MH_SND_TONE_2,              // DTMF for 2 key
    MH_SND_TONE_3,              // DTMF for 3 key
    MH_SND_TONE_4,              // DTMF for 4 key
    MH_SND_TONE_5,              // DTMF for 5 key
    MH_SND_TONE_6,              // DTMF for 6 key
    MH_SND_TONE_7,              // DTMF for 7 key
    MH_SND_TONE_8,              // DTMF for 8 key
    MH_SND_TONE_9,              // DTMF for 9 key
    MH_SND_TONE_A,              // DTMF for A key
    MH_SND_TONE_B,              // DTMF for B key
    MH_SND_TONE_C,              // DTMF for C key
    MH_SND_TONE_D,              // DTMF for D key
    MH_SND_TONE_POUND,          // DTMF for # key
    MH_SND_TONE_STAR,           // DTMF for * key
    MH_SND_NOTE_C4,             //  261.6 Hz       -Piano Notes-
    MH_SND_NOTE_CS4,            //  277.18 Hz
    MH_SND_NOTE_D4,             //  293.6 Hz
    MH_SND_NOTE_DS4,            //  311.1 Hz
    MH_SND_NOTE_E4,             //  329.6 Hz
```

```
MH_SND_NOTE_F4,          //  349.2 Hz
MH_SND_NOTE_FS4,         //  369.9 Hz
MH_SND_NOTE_G4,          //  391.9 Hz
MH_SND_NOTE_GS4,         //  415.3 Hz
MH_SND_NOTE_A4,          //  440.0 Hz
MH_SND_NOTE_AS4,         //  466.1 Hz
MH_SND_NOTE_B4,          //  493.8 Hz
MH_SND_NOTE_C5,          //  523.2 Hz
MH_SND_NOTE_CS5,         //  554.3 Hz
MH_SND_NOTE_D5,          //  587.3 Hz
MH_SND_NOTE_DS5,         //  622.2 Hz
MH_SND_NOTE_E5,          //  659.2 Hz
MH_SND_NOTE_F5,          //  698.5 Hz
MH_SND_NOTE_FS5,         //  739.9 Hz
MH_SND_NOTE_G5,          //  784.0 Hz
MH_SND_NOTE_GS5,         //  830.6 Hz
MH_SND_NOTE_A5,          //  880.0 Hz
MH_SND_NOTE_AS5,         //  932.2 Hz
MH_SND_NOTE_B5,          //  987.7 Hz
MH_SND_NOTE_C6,          // 1046.5 Hz
MH_SND_NOTE_CS6,         // 1108.7 Hz
MH_SND_NOTE_D6,          // 1174.6 Hz
MH_SND_NOTE_DS6,         // 1244.3 Hz
MH_SND_NOTE_E6,          // 1318.5 Hz
MH_SND_NOTE_F6,          // 1397.0 Hz
MH_SND_NOTE_FS6,         // 1479.9 Hz
MH_SND_NOTE_G6,          // 1568.0 Hz
MH_SND_NOTE_GS6,         // 1661.2 Hz
MH_SND_NOTE_A6,          // 1760.0 Hz
```

```
        MH_SND_NOTE_AS6,        // 1864.7 Hz
        MH_SND_NOTE_B6,         // 1975.5 Hz
        MH_SND_NOTE_C7,         // 2093.1 Hz
        MH_SND_NOTE_CS7,        // 2217.4 Hz
        MH_SND_NOTE_D7,         // 2349.3 Hz
        MH_SND_NOTE_DS7,        // 2489.1 Hz
        MH_SND_NOTE_E7,         // 2637.0 Hz
        MH_SND_NOTE_F7,         // 2793.7 Hz
        MH_SND_NOTE_FS,         // 2959.9 Hz
        MH_SND_NOTE_G7,         // 3135.9 Hz
        MH_SND_NOTE_GS7,        // 3322.4 Hz
        MH_SND_NOTE_A7,         // 3520.0 Hz
} MH_mdaToneType;

// Attribution structure of the media handler
typedef enum MH_MdaDevInfo {
        //Support for streaming playback
        MH_MDAINFO_STREAM_PLAY = 0x0001,

        // Uses the transferred buffer contents without copying them
        MH_MDAINFO_CALL_BY_REFERENCE =  0x0002,

        // Media device supporting pause/resume
        MH_MDAINFO_PAUSE_RESUME      = 0x0004,

        //Media handler supporting 'seek'
        MH_MDAINFO_SEEK   = 0x0008,

        // Media device supporting streaming record
```

```
        MH_MDAINFO_STREAM_RECORD      = 0x0010,

        // Left/right sound balance support
        MH_MDAINFO_BALANCE        = 0x0020,

        // Support for mixing support
        MH_MDAINFO_MIXING         = 0x0040,

        //Support for mixing/synchronized playback
        MH_MDAINFO_MIXING_SYNC          =  0x080,

        //Support for non-streaming format recording
        MH MDAINFO RECORD         =0x0100
} MH_MdaDevInfo;

// Media event structure
typedef enum MH_SUB_MEDIA_EVENT {
        MH_MDAEV_MEDIA_EMPTY = 0      // Media handler playback
                                      buffer is empty.
        MH_MDAEV_TONE_EMPTY,          // Tone playback buffer is empty.
        MH_MDAEV_MEDIA_FULL,          // Recording buffer is full.
        MH_MDAEV_MEDIA_ERROR,         // An error has occurred in the
                                      media handler.
        MH_MDAEV_TONE_ERROR,          // An error has occurred in the tone
                                      device.
        MH_MDAEV_OEM_ERROR,           // Media playback or recording at
                                      OEM has been terminated.
        MH MDAEV MEDIA END            //All data in the inner buffer of the
                                      media handler are played.
        MH_MDAEV_MEDIA_STOPPED_AT_TIME //Stopped at the point set by
```

```
        MH_MDACTRL_SET_STOP_TIME
} MH_SUB_MEDIA_EVENT;
// Structure transferring the media event
typedef struct MH_MediaEvent{
        M_Int32 event;  // MH_SUB_MEDIA_EVENT type value
        M_Int32 devID;  // Media handler identifier that has generated the event
        M_Int32 mdaID;// Media handler instance identifier that has generated the
                        event
        M_Int32 size;   // MH_MDAEV_MEDIA_EMPTY,
                        // Given MH_MDAEV_TONE_EMPTY, the size of the data
                        that can be received in the media handler interior buffer
                        // Given MH_MDAEV_MEDIA_FULL, the size of the data
                        recorded in the media handler interior buffer
} MH_MediaEvent;
// Media control command
: Media control command to be used in the MH_mdaControl() function
For the media control command list for supporting each mime type, refer to the
reference item of MH_mdaControl().
typedef enum MH_MdaControl {
        MH_MDACTRL_GET_MEDIA_TIME,      // Obtains the media's current
                                        playback time
        MH_MDACTRL_SET_SYNC,            // Sets the synchronization between
                                        instances
        MH_MDACTRL_GET_SYNC,            // Obtains the synchronized
                                        instance
        NH_MDACTRL_SET_STOP_TIME        // Obtains the stop playback time
        MH_MDACTRL_SET_STOP_TIME,       // Sets the video's playback stop
                                        Time
        MH_MDACTRL_GET_CAPTURE_IMAGE //Captures still images
        MH_MDACTRL_GET_CAPTURE_IMAGE, // Retrieves captured image data
        MH_MDACTRL_PREVIEW_START,       // Starts camera preview
```

```
        MH_MDACTRL_PREVIEW_STOP,        // Stops camera preview
        MH_MDACTRL_SET_MODE             // Sets the mode
}MH_MdaControl;
```

// Media handler control command

: Media handler control command to be used in the MH_mdaDevControl() function

Media handler control command is a command applied to each media handler.

For the media control command list for supporting each mime type, refer to the reference item of MH_mdaDevControl().

```
typedef enum _MH_MdaDevControl {
        // Maximum number of compatible instances
        MH_MDADEVCTRL_GET_INSTANCE_COUNT = 1001,
        // Obtains the camera power status
        MH_MDADEVCTRL_DEVICE_GET_STATUS,
        //Detects the camera device
        MH_MDADEVCTRL_DEVICE_DETECT,
        // // Obtains the camera model name
        MH_MDADEVCTRL_DEVICE_MODEL,
        // Obtains the name list of the mode supported by OEM
        MH_MDADEVCTRL_GET_MODE_LIST }  MH_MdaDevControl;
```

- **Mode Control Command**

Mode refers to the structure composed of the attribution data of each media handler. The terminal manufacturer must be able to support at least one of these modes.

```
typedef enum MH_MdaModeControl {
        MH_MDAMODECTRL_GET,
        MH_MDAMODECTRL_SET
} MH_MdaModeControl;
```

- **Attribution Identifiers Used in Mode Control Command**

: Attribution identifiers used in the MH_mdaModeControl() function

```
typedef enum MH_MdaModePID{
        MH_MDAMODEPID_FORMAT_CODE,
        MH_MDAMODEPID_N_SAMPLE_PER_SEC,
        MH_MDAMODEPID_N_AVG_BYTES_PER_SEC,
        MH_MDAMODEPID_N_CHANNELS,
        MH_MDAMODEPID_N_BIT_PER_SAMPLE,
        MH_MDAMODEPID_BALANCE,
        MH_MDAMODEPID_POSITION_X,
        MH_MDAMODEPID_POSITION_Y,
        MH_MDAMODEPID_WIDTH,
        MH_MDAMODEPID_HEIGHT,
        MH_MDAMODEPID_AXIS,
        MH_MDAMODEPID_BRIGHT,
        MH_MDAMODEPID_MAGPOWER,
        MH_MDAMODEPID_RESOLUTION_X
        MH_MDAMODEPID_RESOLUTION_Y
        MH_MDAMODEPID_YUV_RESOLUTION_X,
        MH_MDAMODEPID_YUV_RESOLUTION Y,
        MH_MDAMODEPID_FRAMERATE,
} MH_MdaModePID;
```

- **MH_mdaTonePlay**

Prototype

M_Int32 MH_mdaTonePlay(MH_mdaToneType tone[], M_Int32 duration[], M_Int32 number, M_Boolean repeat)

Description

This function plays back several tones in order, writes the tone disposition in the O/S's tone playback buffer, and returns the number of copied tones. The MH_MDAEV_TONE_EMPTY event must be transferred to the platform at a suitable period before the tone playback buffer's data is exhausted (suitable period refers to the period wherein there is enough data for the platform to receive the MH_MDAEV_TONE_EMPTY event and copy the data to the tone playback buffer). In case of an error during playback, the O/S must transfer the MH_MDAEV_TONE_ERROR event to the platform. The tone player uses 0 as a media handler identifier; when transferring the MH_MDAEV_TONE_EMPTY event to the platform, it must have the mdaID field of the MH_MediaEvent structure filled with 0. In case of a tone player supporting pause/resume, MH_mdaTonePlay reaches the pause status after copying the data to the buffer in the tone player, and playback must occur during the period when MH_mdaResume is called. If the tone player does not support pause/resume, MH_mdaTonePlay copies the data to the buffer in the tone player, and playback must begin immediately. In case of a tone player that does not support streaming playback, however, an error value is returned when MH_mdaTonePlay is called during playback. The tone player uses 0 as the media handler identifier.

Parameters

[in]	tones	may have the pointer regarding the tone disposition to be played; the defined value of MH_mdaToneType
[in]	duration	Disposition pointer regarding playback duration (unit: ms)
[in]	number	Number of tones to be played
[in]	repeat	TRUE: repeated playback FALSE : play back once

Return Value

Pass

Number of tones existing in the system tone playback buffer

Fail

M_E_INUSE	Already in use
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_mdaFreqTonePlay**

Prototype

M_Int32 MH_mdaFreqTonePlay(M_Int32 hiFreq[], M_Int32 lowFreq[], M_Int32 duration[], M_Int32 number, M_Boolean repeat)

Description

This function plays back several frequency tones in order, stores the frequency tone disposition in the O/S's frequency tone playback buffer, and returns its size. The event (MH_MDAEV_TONE_EMPTY) must be transferred to the platform at a suitable period before the frequency tone playback buffer's data is exhausted (suitable period refers to the period wherein there is enough data for the platform to receive the event (MH_MDAEV_TONE_EMPTY) and copy the data to the frequency tone playback buffer). In case of an error during playback, the O/S must transfer the MH_MDAEV_TONE_ERROR event to the platform. The frequency tone player uses 0 as a media handler identifier; when transferring the event (MH_MDAEV_TONE_EMPTY) to the platform, it must have the mdalD field of the MH_MediaEvent structure filled with 0. In case of a frequency tone player supporting pause/resume, MH_mdaTonePlay reaches the pause status after copying the data to the buffer in the frequency tone player, and playback must occur during the period when MH_mdaResume is called. If the frequency tone player does not support pause/resume, MH_mdaTonePlay copies the data to the buffer in the frequency tone player, and playback must begin immediately. In case of a frequency tone player that does not support streaming playback, an error value is returned when MH_mdaTonePlay is called during playback. The frequency tone player uses 0 as the media handler identifier.

Parameters

[in]	hiFreq	Pointer regarding the high frequency tone structure disposition to be played back
[in]	lowFreq	Pointer regarding the low frequency tone structure disposition to be played back
[in]	duration	Disposition pointer regarding playback duration (unit: ms)
[in]	number	Number of tone structures
[in]	repeat	TRUE: repeated playback FALSE: play back once

Return Value

Pass

Number of tones existing in the system tone playback buffer

Fail

M_E_INUSE	Already in use
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_mdaGetDeviceID**

Prototype

M_Int32 MH_mdaGetDeviceID(M_Char devName)*

Description

This function obtains the identifier of the media handler. The media handler names supported by the O/S can be obtained through MH_sysGetInformation(). The character set obtained through “MEDIADEVICES” from the parameters of the MH_sysGetInformation() function may be used as media handler names. The tone player and frequency tone player use 0 as a media handler identifier; thus, the media handler identifier given by this function must be larger than 0. The name of devices currently supported by WIPI 2.0 platform has been defined in [Table 2-17-1]. In case of a compatible device other than the ones defined below, the device name may be defined at the manufacturer’s discretion. The device names transferred to Parameters have a MIME TYPE format.

Parameters

[in] devName Media handler name verifying compatibility

Return Value

Pass

Media handler identifier

Fail

M_E_NOTSUP Incompatible device name

Side Effects

None

Reference Item

MH_sysGetInformation

[Table 2-17-1] Device names supported on the platform

Mime Type	Description
“Qualcomm_CMX”	Qualcomm CMX
“Yamaha_MA1”	Yamaha MA1
“Yamaha_MA2”	Yamaha MA2
“Yamaha_MA3”	Yamaha MA3 single channel format
“Yamaha_MA5”	
“Yamaha_SMAF”	Yamaha single channel format

“Yamaha_SMAF-Phrase”	Yamaha multi-channel format
“Yamaha_SMAF-Audio”	Yamaha audio format
“audio/MIDI”	MIDI
“audio/WAVE”	WAVE
“audio/MP3”	MP3
“audio/TONE”	Tone
“audio/FREQTONE"	Frequency tone
“IS96”	QCELP-8K
“IS96A”	QCELP-8K
“IS733”	QCELP-13K
“IS127”	EVRC-8K
“G.723.1”	G.723.1
“audio/AAC”	AAC
“audio/AAC+”	AAC+
“video/MPEG4”	Mpeg4
“video/H.263”	H.263
“video/H.264”	H.264
“video/MJPEG”	MJPEG
“image/JPEG”	JPEG

- **MH_mdaGetDeviceInfo**

Prototype

M_Int32 MH_mdaGetDeviceInfo(M_Int32 devID, M_Int32 rtnInfo)*

Description

This function obtains the attribution of the media handler. A device identifier of 0 refers to the frequency tone player or tone player. The attribution supported by the media handler is transmitted to the OR operation value of the bit mask for the Media handler Attribution List Structure (MH MdaDevInfo) as parameter rtnInfo.

1) MH_MDAINFO_STREAM_PLAY bit

: This refers to the compatibility of playback in a media handler-streaming mode. This means that MH_mdaWriteData (given a device identifier of 0, MH_mdaTonePlay, MH_mdaFreqTonePlay function) can have new data copied to the internal buffer of the media handler during media playback. Likewise, the media handler must be able to play back new data in succession. If streaming playback is supported, MH_MDAINFO_CALL_BY_REFERENCE bit may not be installed.

2) MH_MDAINFO_CALL_BY_REFERENCE bit

: The media handler does not copy the data buffer transferred to MH_mdaWriteData (given a device identifier of 0, MH_mdaTonePlay, MH_mdaFreqTonePlay) in the interior buffer; instead, it uses the data buffer directly. If this bit is not set up, the data to be transferred must be copied to the interior buffer for use.

3) MH_MDAINFO_PAUSE_RESUME bit

: Refers to the media handler supporting the pause/resume feature

4) MH_MDAINFO_SEEK bit

: Refers to the media handler supporting the seek feature

5) MH_MDAINFO_STREAM_RECORD bit

: This refers to the media handler supporting streaming record (audio/video). This means that the recorded data can be copied to the platform’s buffer from the buffer within the recording device during recording through MH_mdaCopy. The recording device must be able to record data in succession in the empty buffer.

6) MH_MDAINFO_BALANCE bit

: This refers to the media handler supporting the left/right sound balance feature. With 50 as the basic setup, 0 will have only the left sound activated, and 100, only the right sound.

7) MH_MDAINFO_MIXING bit

: This refers to the media handler supporting multiple media data playback. When

multiple playback is not supported, and various media handler instances are being created to have the same type of media handler, a particular error (M_E_INPROGRESS) should be returned.

8) MH_MDAINFO_MIXING_SYNC bit

: This refers to the multi-channel synchronized playback feature. Thus, it is set up in each channel of the media handler when playback is enabled with various synchronized files.

9) MH_MDAINFO_RECORD bit

This refers to the support to recording by a type other than streaming. This cannot copy audio/video recorded data from the inner buffer of the media handler to the buffer using MH mdaCopy during audio/video recording. Data copying is possible only when audio/video recording is completed.

Parameters

[in]	devID	Media handler identifier returned from MH_mdaGetDeviceID()
[out]	rtnInfo	OR operation value of the bit mask for the Media handler Attribution Structure (MH MdaDevInfo)

Return Value

Pass

0

Fail

M_E_ERROR	Error

Side Effects

None

Reference Item

None

- **MH_mdaOpenDevice**

Prototype

M_Int32 MH_mdaOpenDevice(M_Int32 devID, M_char param)*

Description

This function resets and opens the media handler related to the media handler identifier passed on to parameters as a default value. It then creates and returns the instance's identifier from the media handler. Here, note that the identifier of the media handler instance is related to the media handler identifier passed on to parameters. This is because each API can operate through each media handler only when a particular media handler identifier is recognized, whereas the identifier of the media handler instance is used to approach the media HAL API such as MH_mdaPlay(), MH_mdaPause(), etc. The media handler must have at least a single media handler instance; if the media handler supports synchronized playback, it may have more than one media handler instance.

The necessary parameters can be transferred when opening the device.

For the camera, the power must be turned on using this function.

Parameters

[in]	devID	Media handler identifier returned from MH_mdaGetDeviceID()
[in]	param	These are the necessary parameters when opening the media handler. In case the parameters are not needed, a NULL value is inputted. The beginning keyword in the string passed on from the parameters may differ according to the parameters requested by the device (refer to the table below). These parameters do not necessarily refer to the attribution data of the media handler. When opening the media handler, the attribution value set as a default value at the terminal is used; attribution data can be changed using the MH_mdaControl function (refer to the reference item for details).

Return Value

Pass

Media handler instance identifier

Fail

M_E_ERROR	When an error has occurred
M_E_NOTSUP	Incompatible media handler
M_E_INPROGRESS	When the maximum instance number has been exceeded

Side Effects

None

Reference Item

1. Media that can directly access and handle streaming media in URL format. Add "-streamURL" in front of the URL string as a device prefix prior to transmission.

Ex 1) –streamURL http://www.media.com/m.mp3

Ex 2) –streamURL mms://www.media.com/m.mp3

Ex 3) –streamURL rtsp://www.media.com/m.mp3

2. Media handler that can directly access and play the file

Add "-file" in front of the file name as a prefix prior to transmission.

Ex 1) –file media/sample.mp3

Ex 2) –file sample.mov

- **MH_mdaCloseDevice**

Prototype

M_Int32 MH_mdaCloseDevice(M_Int32 mdaID)

Description

This function terminates the resource of the media handler instance related to the media handler instance identifier passed on from the parameters. The media handler is terminated if the instance of a particular media handler no longer exists. If this function is called in an already terminated device, it does not play any part and returns a success value. This function must succeed.

This function should also be used to turn off a camera.

Parameters

[in] mdaID Media handler instance identifier obtained in MH_mdaOpenDevice()

Return Value

Pass

When the device is closed successfully or has already been closed, the value is 0.

Fail

M_E_INVALID	Invalid media handler instance identifier
M_E_ERROR	Other errors

Reference Item

MH_mdaOpenDevice

- **MH_mdaWriteData**

Prototype

*M_Int32 MH_mdaWriteData(M_Int32 mdaID, void *buf, M_Int32 size)*

Description

This function copies the data to be played back in the media handler interior buffer. The media handler must transfer the event (MH_MDAEV_MEDIA_EMPTY) to the platform at a suitable period before the interior buffer is exhausted (suitable period refers to the period wherein there is enough data for the platform to receive the event (MH_MDAEV_MEDIA_EMPTY) and copy the data to the media handler interior buffer). If MH_mdaStop() is not executed, and device interior buffer is exhausted, the media handler must dispose of it (mute, etc.) accordingly. When trying to copy a larger amount of data than what the device interior buffer can accommodate, the remainder is returned after copying depending on the device interior buffer's capacity. In case of an error during playback, the MH_MDAEV_MEDIA_ERROR event is transferred to the platform. If the media handler does not support streaming, an error is returned; otherwise, in case streaming is supported, MH_mdaWriteData is called.

Parameters

[in]	mdaID	Media handler instance identifier obtained from MH_mdaOpenDevice()
[in]	buf	Data buffer
[in]	size	Length to be copied

Return Value

Pass

Size copied to the media interior buffer

Fail

M_E_INUSE	Already in use
M_E_ERROR	Other errors
M_E_NOTSUP	Media handler does not support playback.

Side Effects

None

Reference Item

None

- **MH_mdaPlay**

Prototype

M_Int32 MH_mdaPlay(M_Int32 mdaID, M_Boolean repeat)

Description

The media handler begins playback in the background mode when the MH_mdaPlay() function is called. In case of playback of straming type, if a certain size is exhausted in the interior buffer, the event (MH_MDAEV_MEDIA_EMPTY) is transferred to the platform before the buffer is completely exhausted to enable playable data to be copied to the media handler. In case of an error during playback, the MH_MDAEV_MEDIA_ERROR event must be transferred to the platform. If the playback speed in the media handler is faster than the copying speed from the platform to the media handler, the buffer can be exhausted; after data copying is resumed, however, it must operate normally.

Parameters

[in]	mdaID	Media handler instance identifier obtained from MH_mdaOpenDevice()
[in]	repeat	TRUE: repeated playback FALSE: play back once

Return Value

Pass

0

Fail

M_E_INPROGRESS	When the media handler is already in use
M_E_ERROR	Other errors
M_E_NOTSUP	Media handler does not support playback.

Side Effects

None

Reference Item

None

- **MH_mdaPause**

Prototype

M_Int32 MH_mdaPause(M_Int32 mdaID)

Description

This function pauses the media in playback. If MH_mdaResume is called in the pause mode, the interior state must be maintained to enable playback to resume. If the media handler does not play back when this function is called, An M_E_ERROR value must be returned; if it does not support pause/resume, an error value (M_E_NOTSUP) is returned.

Parameters

[in]	mdaID	Media handler instance identifier obtained from MH_mdaOpenDevice()

Return Value

Pass

0

Fail

M_E_NOTSUP	Media handler does not support pause/resume.
M_E_ERROR	Called while the media handler is not in the playback mode

Side Effects

None

Reference Item

None

- **MH_mdaResume**

Prototype

M_Int32 MH_mdaResume(M_Int32 mdaID)

Description

This function resumes media playback that has been paused. If this function is not called in the pause mode, M_E_ERROR must be returned; otherwise, if pause/resume is not supported, an error value (M_E_NOTSUP) is returned.

Parameters

[in]	mdaID	Media handler instance identifier obtained from MH_mdaOpenDevice()

Return Value

Pass

0

Fail

M_E_NOTSUP	Media handler does not support pause/resume.
M_E_ERROR	Called while the media handler is not in the pause mode.

Side Effects

None

Reference Item

None

- **MH_mdaSeek**

Prototype

M_Int32 MH_mdaSeek(M_Int32 mdaID, M_Int32 seekTime)

Description

This function designates the playback starting point in millisecond. M_E_ERROR is returned if this function is not called in the pause mode; the same is true when the media handler does not support seek.

Parameters

[in]	mdaID	Media handler instance identifier obtained from MH_mdaOpenDevice()
[in]	seekTime	Playback point time (ms); a value smaller than 0 is set as the starting point; otherwise, a value larger than the entire playback of the data is set as the ending point

Return Value

Pass

0

Fail

M_E_NOTSUP	Media handler does not support seek.
M_E_ERROR	Called while the media handler is not in the pause mode

Side Effects

None

Reference Item

None

- **MH_mdaStop**

Prototype

M_Int32 MH_mdaStop(M_Int32 mdaID)

Description

This function stops playback/recording. Regardless of the state of the media handler, media playback is stopped when this function is called. It does not play a part and return a success value when called in a stopped media. This function must always succeed.

Parameters

[in]	mdaID	Media handler instance identifier obtained from MH_mdaOpenDevice()

Return Value

Pass

In playback mode or stopped state, 0

When in recording mode, the size of data recorded up to that point in the interior buffer

Fail

M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_mdaGetVolume**

Prototype

M_Int32 MH_mdaGetVolume (M_Int32 devID)

Description

This function reads the volume value from the volume source. Media handlers that support volume setup read the volume value in each media handler; those that do not support volume setup point to the same volume source even when the media handler identifier is different. The minimum value of volume is 0, and the maximum value, 100. The returned volume value must be converted into a value between 0 and 100. The value between 0 and 100 coinciding with the volume level follows the percentage of volume level supported by the hardware as in the example below. The levels of volume supported by the hardware are returned in MH_sysGetInformation().

Ex.) Hardware with two volume levels => 1-50: small volume, 51-100: large volume

Hardware with three volume levels => 1-33: small volume, 34-66: medium volume, 67-100: large volume

Parameters

[in]	devID	Media handler identifier with its return value returned from MH_mdaGetDeviceID()

Return Value

Pass

Volume value

Fail

M_E_NOTSUP	Media handler without volume value

Side Effects

None

Reference Item

None

- **MH_mdaSetVolume**

Prototype

M_Int32 MH_mdaSetVolume (M_Int32 devID, M_Int32 value)

Description

This function designates the value at the volume source. A volume value that is smaller than the minimum volume is set to minimum volume and vice versa. The minimum volume value is 0, and the maximum value, 100. Media handlers with volume setups can do so with each media handler, whereas those without volume setup may point to the same volume source despite different media handler identifiers.

Parameters

[in]	devID	Media handler with its return value returned from MH_mdaGetDeviceID()
[in]	value	Volume value (between 0-100)

Return Value

Pass

0

Fail

M_E_NOTSUP	Media handler without volume value

Side Effects

None

Reference Item

None

- **MH_mdaControl**

Prototype

M_Int32 MH_mdaControl(M_Int32 mdaID, M_Int32 cmd, void buf1, void* buf2)*

Description

This function executes the control command in the media handler instance.

For media handler instances, there may be cases wherein new features other than the ones provided by HAL should be used. New features that are not defined at HAL may be added to the media handler with existing features, or when a new media handler is added to the platform. In this case, new features suggested by the vendor may have to be added as user-defined functions. This function allows the addition of new user-defined functions that are not defined at HAL.

Compatible media control commands according to each mime type are as follows:

[Table 2-17-2] Compatible media control command according to each MIME TYPE

Mime Type	Compatible Media Control Command
“Qualcomm_CMX”	MH_MDACTRL_GET_MEDIA_TIME, // Media’s current playback time MH_MDACTRL_SET_SYNC, // Synchronization setup between instances MH_MDACTRL_GET_SYNC, // Obtaining synchronized instances MH_MDACTRL_GET_STOP_TIME // Retrieves the stop time MH_MDACTRL_SET_STOP_TIME // Sets up the stop time MH_MDACTRL_SET_MODE // Sets up mode with name
“Yamaha_MA1”	
“Yamaha_MA2”	
“Yamaha_MA3”	
“Yamaha_MA5”	
“Yamaha_SMAF”	
“Yamaha_SMAF-Phrase”	
“Yamaha_SMAF-Audio”	
“audio/MIDI”	
“audio/WAVE”	
“audio/MP3”	
“audio/TONE”	
“audio/FREQTONE"	
“IS96”	
“IS96A”	
“IS733”	
“IS127”	
“G.723.1”	
“audio/AAC”	
“audio/AAC+”	

“video/MPEG4” “video/H.263” “video/H.264” “video/MJPEG” “image/JPEG”	MH_MDACTRL_GET_MEDIA_TIME, // Media’s current playback time MH_MDACTRL_PREVIEW_START, // Starts camera preview MH_MDACTRL_PREVIEW_STOP, // Ends camera preview MH_MDACTRL_GET_STOP_TIME, // Retrieves the stop time MH_MDACTRL_SET_STOP_TIME // Sets up the stop time MH_MDACTRL_CAPTURE_IMAGE // Captures still images MH_MDACTRL_GET_CAPTURE_IMAGE, // Retrieves the captured still image data MH_MDACTRL_SET_MODE // Sets up mode with name

Parameters

[in]	mdaID	Device instance identification returned from MH_mdaOpenDevice()
[in]	cmd	Control command
[in]	buf1	buf1 usable at the control command
[in/out]	buf2	buf2 usable at the control command

Return Value

Pass

0

Fail

M_E_ERROR	Incompatible command, or failure in executing command

Side Effects

None

Reference Item

[Table 2-17-3] Description of media control command and parameters

Cmd	MH_MDACTRL_GET_MEDIA_TIME
buf1	None
buf2	[out] Pass: *(M_int32*) buf2 = Current playback time (unit: millisecond) Fail: *(M_Int32*) buf2 = M_E_ERROR Other errors *(M_Int32*) buf2 = M_E_NOTSUP Incompatible
Description	Obtains current playback time (unit: millisecond) vis-à-vis the entire playback time
Remarks	
cmd	MH_MDACTRL_SET_SYNC
buf1	[in] *(M_Int32*) buf1[0] = Disposition size of the media handler instance identifier *(M_Int32*) buf1[1] = First slave media instance identifier to be synchronized *(M_Int32*) buf1[2] = Second slave media instance identifier to be synchronized …… Repeat for the size of the disposition
buf2	[out] Pass: *(M_int32*) buf2 = 0 Fail: *(M_Int32*) buf2 = M_E_ERROR Other errors *(M_Int32*) buf2 = M_E_NOTSUP Incompatible
Description	Sets up channel synchronization between the media played back at multi-channels Synchronization termination is terminated by turning over 0 for the disposition size of the *(M_Int32*) buf1[0] = media handler instance identifier
Remarks	
cmd	MH_MDACTRL_GET_SYNC
buf1	[in] *(M_Int32*) buf1 = Maximum size of the multi-channel disposition
buf2	[out] Pass: *((M_Int32*)buf2+0) = First slave media instance identifier synchronized *((M_Int32*)buf2+1) = First slave media instance identifier synchronized …Repeat for the size of the disposition Fail: *(M_Int32*)buf2 = M_E_NOTSUP: Does not support synchronization *(M_Int32*)buf2 = M_E_ERROR: Other errors
Description	Obtains channel synchronization data between the playback media at

	multi-channels
Remarks	
cmd	MH_MDACTRL_SET_STOP_TIME (unit: millisecond)
buf1	[in] *(M_Int32*) buf1 = Playback ending point (in millisecond)
buf2	[out] Pass: *(M_Int32*) buf2 = 0 Fail: *(M_Int32*) buf2 = M_E_NOTSUP: Incompatible *(M_Int32*) buf2 = M_E_ERROR: Other errors
Description	Sets the playback ending point vis-à-vis the media's entire playback time
Remarks	
cmd	MH_MDACTRL_CAPTURE_IMAGE
buf1	[in] (M_Char*) buf1 = Buffer for storing captured screen shot
buf2	[in] *(M_Int32*) buf2 = Buffer size for storing captured screen shot
Description	Captures the screen shot of the video being played back
Remarks	Return value Pass: Size of the captured screen shot Fail: M_E_NOTSUP: Incompatible M_E_ERROR: Other errors
cmd	MH_MDACTRL_PREVIEW_START
buf1	None
buf2	[out] Pass: *(M_Int32*) buf2 = 0 Fail: *(M_Int32*) buf2 = M_E_ERROR Other errors
Description	Begins preview playback according to the currently set screen mode and screen size; if preview is already being played back, nothing happens
Remarks	
cmd	MH_MDACTRL_PREVIEW_STOP
buf1	None
buf2	None
Description	Stops preview playback during the playback mode; if preview is not in the playback mode, nothing happens
Remarks	This function must always succeed.
cmd	MH_MDACTRL_SET_MODE
buf1	[in] *(M_Char*) buf1: Mode name
buf2	[out] Pass: *(M_Int32*) buf2 = 0 Fail: *(M_Int32*) buf2 = M_E_ERROR Other errors

	(M_Int32) buf2 = M_E_NOTSUP Incompatible mode name *(M_Int32*) buf2 = M_E_INVALID Invalid mode name
Description	Sets the mode as the mode name passed to buf1
Remarks	
cmd	MH_MDACTRL_GET_STOP_TIME
buf1	[out] (M_Int32*) buf1 : Currently set stop time
buf2	None
Description	Transmits the currently set temporary stop time in millisecond When the temporary stop time is not set, transmit -1 as parameter buf1.
Remarks	

- **MH_mdaDevControl**

Prototype

M_Int32 MH_mdaDevControl(M_Int32 devID, M_Int32 cmd, void buf1, void* buf2)*

Description

This function executes the command control in the media handler.

For media handler instances, there may be cases wherein new features other than the ones provided by HAL should be used. New features that are not defined at HAL may be added to the media handler with existing features, or when a new media handler is added to the platform. In this case, the new features suggested by the vendor may have to be added as user-defined functions. This function allows the addition of new user-defined functions that are not defined at HAL.

This function is used if the control command must be executed by each media handler; otherwise, if the control command must be executed by each media handler instance, MH_mdaControl() is used. The control command executed by each media handler may include those used to switch the camera on or off, whereas that executed by each media handler instance includes the current playback time of the media contents. The supportable media handler control command per mime type is as follows:

[Table 2-17-4] Compatible media handler control command with each MIME TYPE

Mime Type	Compatible Media handler Control Command
“Qualcomm_CMX”	MH_MDADEVCTRL_GET_INSTANCE_COUNT, // Maximum number of supportable instances MH_MDADEVCTRL_GET_MODE_LIST // Obtains the name list of the supporting mode at OEM
“Yamaha_MA1”	
“Yamaha_MA2”	
“Yamaha_MA3”	
“Yamaha_MA5”	
“Yamaha_SMAF”	
“Yamaha_SMAF-Phrase”	
“Yamaha_SMAF-Audio”	
“audio/MIDI”	
“audio/WAVE”	
“audio/MP3”	
“audio/TONE”	
“audio/FREQTONE"	
“IS96”	
“IS96A”	

“IS733”	
“IS127”	
“G.723.1”	
“audio/AAC”	
“audio/AAC+”	
“video/MPEG4”	
“video/H.263”	
“video/H.264”	
“video/MJPEG”	MH_MDADEVCTRL_GET_INSTANCE_COUNT, // Maximum number of compatible instances MH_MDADEVCTRL_DEVICE_GET_STATUS, // Determines the camera’s power status MH_MDADEVCTRL_DEVICE_DETECT, // Detects the camera device MH_MDADEVCTRL_DEVICE_MODEL, // Acquires the camera’s model name MH_MDADEVCTRL_GET_MODE_LIST // Obtains the name list of the compatible mode at OEM
‘image/JPEG”	

Parameters

[in]	devID	Device identifier as the return value of MH_mdaGetDeviceID()
[in]	cmd	Control command
[in]	buf1	buf1 that can be used at the control command
[in/out]	buf2	buf2 that can be used at the control command

Return Value

Pass

0

Fail

M_E_ERROR	Incompatible command, or failure in command execution

Side Effects

None

Reference Item

[Table 2-17-5] Description of the media handler control command and parameters

cmd	MH_MDADEVCTRL_GET_INSTANCE_COUNT
buf1	None
buf2	[out] *(M_Int32*) buf2: Number of compatible instances
Description	Obtains the number of compatible instances in this device
Remarks	Must support at least one device
cmd	MH_MDADEVCTRL_DEVICE_GET_STATUS
buf1	None
buf2	[out] *(M_Boolean*) buf2: TRUE: Media handler power is on *(M_Boolean*) buf2: FALSE: Media handler power is off
Description	Obtains the power status of the device
Remarks	
cmd	MH_MDADEVCTRL_DEVICE_DETECT
buf1	None
buf2	[out] Pass: *(M_Int32*) buf2 = 0 Fail: *(M_Int32*) buf2 = M_E_ERROR
Description	Detects the built-in/exterior camera
Remarks	
cmd	MH_MDADEVCTRL_DEVICE_MODEL
buf1	[in] *(M_Int32*) buf1: Length of the camera model name
buf2	[out] Pass: (M_Char*) buf2 = Camera model name Fail: *(M_Int32) buf2 = M_E_LONGNAME When the model name is too long *(M_Int32) buf2 = M_E_ERROR Model name unidentifiable
Description	Obtains the camera's model name
Remarks	
cmd	MH_MDADEVCTRL_GET_MODE_LIST
buf1	Size of the buffer used to obtain the name list of the mode supported by the manufacturer
buf2	[out] Pass: (M_Char*) buf2 = Name list of the mode supported by the manufacturer Fail: *(M_Int32) buf2 = M_E_ERROR Other errors *(M_Int32) buf2 = M_E_SHORTBUF Buffer size too short
Description	Obtains the name list of the mode supported by the manufacturer. Mode refers to the structure composed of the general attribution data of the media handler and utilizes the MH_mdaModeControl() function

	when obtaining or adjusting the mode structure's value. The general attributions per device are already defined, although they may have other specific attributions added according to the telecommunications provider or manufacturer. If various modes are supported, "," is inserted between the mode names to differentiate them. Ex.: "DEFAULT_MODE, SKT_MODE, LG_MODE"
Remarks	At least one mode should be supported (name: "DEFAULT_MODE").

- **MH_mdaModeControl**

Prototype

M_Int32 MH_mdaModeControl(M_Int32 mdaID, M_Char modeName, MH_MdaModeControl cmd, MH_MdaModePID pID, void* buf)*

Description

The name list of the media handler's mode supported by the current manufacturer may be obtained using the MH_MDADEVCTRL_GET_MODE_LIST command control of MH_mdaDevControl. Mode refers to the structure with abstracted contents possessed by the media handler. The introduction of such definition enables convenience of application development for the Contents Provider using the media handler by designating all the attribution factors of the media handler in one attempt using the mode defined by the manufacturer or telecommunications provider. The terminal must have at least one compatible mode named "DEFAULT_MODE." Aside from the "DEFAULT_MODE," other modes provided by the manufacturer or telecommunications provider may have the attribution factor within the "DEFAULT_MODE" or have an entirely new attribution data added. Attribution values of the mode named "DEFAULT_MODE" are set to default, can be read and modified. The attribution data values of other modes supported and separately defined by the manufacturer or telecommunications provider may set the authority for read or write function by itself. For the attribution values of the modes supported by the manufacturer or telecommunications provider, check with the manufacturer or telecommunications provider. The MH_mdaModeControl() function enables reading the attribution values in these modes possible; even writing is enabled with the attribution value of the "DEFAULT_MODE." For attribution data other than the mode named "DEFAULT_MODE," if the authority for writing function has been given by the manufacturer or telecommunications provider, writing is enabled only for that particular attribution data. The general attribution values of each media handler are already defined in the table below. The manufacturer or telecommunications provider may add attribution. Note that the adjusted value is directly applied when adjusting the attribution value of the mode name currently designated; when adjusting the attribution value of another mode name currently not set up, however, the adjusted values are not applied immediately, but done so when MH_mdaControl()'s MH_MDACTRL_SET_MODE command control is called with the mode name as the Parameters.

Parameters

[in]	mdaID	Instance identifier of device media that is the return value of MH_mdaOpenDevice()
[in]	modeName	Mode name: the mode name list supported by the terminal may be obtained by MH_mdaDevControl's MH_MDADEVCTRL_GET_MODE_LIST command control, which refers to the mode name trying to obtain or adjust the attribution data value from the mode's name list.
[in]	cmd	control command MH_MDAMODECTL_GET/MH_MDAMODECTL_SET
[in]	pID	Attribution ID for executing the control command
[in/out]	buf	buf used at the control command If the control command is MH_MDAMODECTL_SET, then [in] If the control command reads MH_MDAMODECTL_GET, then [out]

Return Value

Pass

0

Fail

M_E_ERROR	Incompatible command, or failure in command execution

Side Effects

None

Reference Item

1. Mode Name Regulations

All modes have names to distinguish themselves from other modes. Mode names are not decided at the developer's discretion; rather, they are decided in accordance with the following regulations:

① Must have at least one compatible mode, and its name should be "DEFAULT_MODE."

② In case of a mode supported by the manufacturer or telecommunications provider, the regulation below is followed:

(Manufacturer, telecommunications provider name)_MODE_(Index)

Ex.) If SKT supports three modes, each mode name will be: SKT_MODE_0, SKT_MODE_1, and SKT_MODE_2.

2. DEFAULT_MODE's attribution factor

A mode with the name "DEFAULT_MODE" is one that has to be provided unconditionally at the terminal. This mode is composed of attribution factors generally possessed by each media handler.

3. Attribution factor in the DEFAULT_MODE per media type

[Table 2-17-6] Attribution factor in the DEFAULT_MODE per media type

Media handler		Attribution Data
	"audio/MIDI"	Balance Tempo Pitch
	"audio/TONE"	
	"audio/FREQTONE"	
	"Qualcomm_CMX"	
	"Yamaha_MA1"	Balance
	"Yamaha_MA2"	
	"Yamaha_MA3"	
	"Yamaha_MA5"	
	"Yamaha_SMAF"	
	"Yamaha_SMAF-Phrase"	
	"Yamaha_SMAF-Audio"	
Vocoder	"IS96"	Balance
	"IS96A"	
	"IS733"	
	"IS127"	
	"G.723.1"	
	"AMR-WB"	
	"AMR-NB"	
General sound	"audio/WAVE"	Sample per second Significant bits per sample Number of channels Balance
	"audio/MP3"	
	"audio/AAC"	
	"audio/AAC+"	
Video	"video/MPEG4"	Location (x position, y position) Size (width, height) Axis Bright
	"video/H.263"	
	"video/H.264"	
Video	"video/MJPEG"	

capture	‘image/JPEG”	MagPower Resolution (x , y) YUV Resolution (x , y) FrameRate

4. Detailed description of the attribution ID in the DEFAULT_MODE

[Table 2-17-7] Detailed description of the attribution ID in the DEFAULT_MODE

Name of Attribution	Description
Tempo	Tempo when the media is midi
Pitch	Pitch when the media is midi
Sample per second	Audio sample baud rate; the default value is 8000KHz
Significant bits per sample	Audio sample size; the default value is 8 bits
Number of channels	Number of channels (mono/stereo); the default value is mono
Balance	This refers to the sound balance; a value smaller than 50 will have a bigger left sound, whereas a value larger than 50 will have the opposite effect. The balance value domain is between 0 and 100. The default value is 50.
Location (x position, y position)	X and y coordinates of the screen (in pixels); the default value for x and y coordinates is 0
Size (width, height)	Screen width and height (in pixels); the default value is the size of the entire screen
Axis	Screen rotation/inversion value (refer to MH_MdaCameraSetAxis); the default value is a normal screen
Bright	Screen brightness (percentage unit); the default value is 50
MagPower	Screen magnifying power (percentage unit; 100 for regular magnification, 200 for double, 400 for quadruple, 150 for 1.5 times magnification); the default value is 100
Resolution (x , y)	X, Y values of resolution (in pixels); the default value is 320*240
YUV Resolution (x , y)	X, Y values of the YUV resolution (in pixels); the default value is 320*240
FrameRate	Frames per second (default value is 10)

- **MH_mdaRecord**

Prototype

M_Int32 MH_mdaRecord (M_Int32 mdaID)

Description

This function starts recording of sound and image. If the MH_mdaRecord() function is called, the media handler begins recording in the background and copies the sampled data to the media handler interior buffer. If a certain amount is filled at the interior buffer, the event (MH_MDAEV_MEDIA_FULL) is transferred to the platform before the buffer becomes full; thus copying the data recorded on the platform. In case of an error during recording, the MH_MDAEV_MEDIA_ERROR event must be transferred to the platform. If the copying speed of the recorded data on the platform is slower than the recording speed of the media handler, the interior buffer may become full. In case of a full interior buffer, data is recorded until data copied to the platform is scrapped. A full interior buffer within a device that does not support streaming recording is stopped.

Parameters

[in] mdaID Device instance identifier returned from MH_mdaOpenDevice()

Return Value

Pass

0

Fail

M_E_INPROGRESS	When the media handler is already in use
M_E_ERROR	Other errors
M_E_NOTSUP	Media handler does not support recording.

Side Effects

None

Reference Item

None

- **MH_mdaCopy**

Prototype

M_Int32 MH_mdaCopy(M_Int32 mdaID, void buf, M_Int32 size)*

Description

This function copies the recorded audio/video data in the media handler interior buffer. If the device does not support streaming recording, an error value is returned, whereas MH_mdaCopy is called in recording.

Parameters

[in]	mdaID	Device instance identifier returned from MH_mdaOpenDevice()
[out]	buf	Buffer where recorded data is to be copied
[in]	size	Size to be copied

Return Value

Pass

Actual copied size

Fail

M_E_ERROR	Called while recording at a device that does not support streaming recording
M_E_NOTSUP	Media handler does not support recording.

Side Effects

None

Reference Item

None

- **MH_mdaSetMuteState**

Prototype

*M_Int32 MH_mdaSetMuteState(M_Char *strCategory, M_Boolean bmute)*

Description

This function sets the mute state per volume category of the terminal.

For the volume category supported by the terminal, refer to [Table 2-17-8] in the reference item.

Parameters

[in] strCategory Category string of the default value

[in] bmute Mute setup

TRUE Mute mode

FALSE Sound mode

Return Value

Pass

M_E_SUCCESS Pass

Fail

M_E_ERROR Fail

Side Effects

None

Reference Item

[Table 2-17-8] Volume category supported by the terminal

Category String	Description	Default Volume Example
GENERAL	Used for general applications	Application volume level of the terminal
VOICE	Voice Play/Record feature	Calling volume of the terminal
RING	This is the ringing feature, e.g., in case the current ringing mode is set to vibration, vibration occurs during playing instead of sounds. In case of a separate melody speaker, sounds occur through it. It is the same as the feature of a terminal receiving a call.	Ringing volume of the terminal

KEY	Key tone feature	Key tone volume of the terminal
MESSAGE	SMS message reception alert feature	SMS message volume
ALARM	Alarm alert feature	Alarm volume
ALERT	Various alert features, e.g., no service, low battery, etc.	Alert volume
MMEDIA	This is the volume for the multimedia device. It indicates the master volume of all multimedia devices supported by the platform. The master volume affects all multimedia devices. To set the volume by media device, use the MH_mdaGetVolume/MH_mdaSetVolume function.	Multimedia volume
GAME	Feature used when playing a game	Game volume
OEM	Used when setting a volume level other than those defined above	Other volume

- **MH_mdaGetMuteState**

Prototype

*M_Boolean MH_mdaGetMuteState(M_Char *strCategory)*

Description

This function obtains the mute setup status per terminal's volume category.

For volume categories supported by the terminal, refer to [Table 2-17-8] in the reference item of the MH_mdaSetMuteState function.

Parameters

[in] strCategory Category string of the default volume

Return Value

TRUE

Mute mode

FALSE

Sound mode

Side Effects

None

Reference Item

None

- **MH_mdaGetDefaultVolume**

Prototype

*M_Int32 MH_mdaGetDefaultVolume(M_Char *strCategory)*

Description

This function obtains the default volume set by terminal.

The categories of the default volume supported by terminal are shown in the table below (this classification can vary by terminal manufacturer or terminal model; for volume category supported by terminal, refer to [Table 2 -17-9] of the reference item):

Parameters

[in] strCategory Category string of the default volume

Return Value

Pass

Volume value (between 0-100)

Fail

M_E_NOTSUP	Issued when the default volume setting unsupported
M_E_ERROR	Issued in case of other error
M_E_INVALID	Issued in case of invalid parameter

Side Effects

None

Reference Item

[Table 2-17-9] Volume category supported by the terminal

Category ID	Description	Default Volume Example
MH_MDAVOLCATE_VOICE	Voice Play/Record feature	Calling volume of the terminal
MH_MDAVOLCATE_RING	This is the ringing feature, e.g., in case the current ringing mode is set to vibration, vibration occurs during playing instead of sounds. In case of a separate melody speaker, sounds occur through it. It is the same as the feature of a terminal receiving a call.	Ringing volume of the terminal
MH_MDAVOLCA	Key tone feature	Key tone volume of

TE_KEYTONE		the terminal
MH_MDAVOLCATE_MESSAGE	SMS message reception alert feature	SMS message volume
MH_MDAVOLCATE_ALARM	Alarm alert feature	Alarm volume
MH_MDAVOLCATE_ALERT	Various alert features, e.g., no service, low battery, etc.	Alert volume
MH_MDAVOLCATE_MMEDIA	This is the volume for the multimedia device. It indicates the master volume of all multimedia devices supported by the platform. The master volume affects all multimedia devices. To set the volume by media device, use the MH_mdaGetVolume/MH_mdaSetVolume function.	Multimedia volume

- **MH_mdaSetDefaultVolume**

Prototype

*M_Int32 MH_mdaSetDefaultVolume(M_Char *strCategory, M_Int32 value)*

Description

This function sets the default volume on the terminal. The categories of the default volume supported by terminal are as follows (this classification can vary by terminal manufacturer or terminal model; for the volume category supported by terminal, refer to [Table 2-17-9] in the reference item of the MH_mdaGetDefaultVolume function):

Parameters

[in]	strCategory	Default volume category string
[in]	value	Volume value (between 0-100)

Return Value

Pass

0

Fail

M_E_NOTSUP	Issued when the default volume setting is unsupported
M_E_ERROR	Issued in case of other error
M_E_INVALID	Issued in case of invalid parameter

Side Effects

None

Reference Item

None

2.18. SMS

Standard for sending a message using SMS

2.18.1. Related Data Types

```
// SMS event
typedef enum MH_SUB_SMS_EVENT {
        MH_SMSEV_SEND_NOTIFY    // Transferred when message is transmitted
} MH_SUB_SMS_EVENT;


// Structure transferring the SMS event
typedef struct MH_SMSEvent {
M_Int32 event;  // MH_SUB_SMS_EVENT value
M_Int32 param1;         // 1 in case of success given MH_SMSEV_SEND_NOTIFY
//(Referring to one message that has been transmitted); -1 when failing to transmit
} MH_SMSEvent;
```

- **MH_smsSend**

Prototype

M_Int32 MH_smsSend(M_Char telnum, M_Byte* smsMsg, M_Int32 len)*

Description

This function transfers regular short messages. The transmitter and transmitting time are managed by HAL.

A return value of M_E_WOULDBLOCK means that data cannot be transmitted immediately due to the system interior. In this case, the event must be transferred to the platform in the order of callups on transmission success after processing the data.

Parameters

[in]	telnum	Party's telephone number
[in]	smsMsg	Message to be transmitted
[in]	len	Length of message

Return Value

Pass

0

Fail

M_E_ERROR	Other errors
M_E_WOULDBLOCK	When data cannot be transmitted directly due to system interior

Side Effects

None

Reference Item

None

- **MH_smsGetMaxMsgLength**

Prototype

M_Int32 MH_smsGetMaxMsgLength(void)

Description

Returns the maximum length of transmittable messages converted in byte units

Parameters

None

Return Value

Pass

Maximum value of the message length in bytes

Fail

M_E_ERROR	Unknown error

Side Effects

None

Reference Item

None

2.19. Location Information

Location information API only needs HAL API vis-à-vis GPS because it can bring the base station location info through the system property at API or HAL containing the GPS info and base station location info simultaneously. GPS-related APIs have been prepared with Qualcomm, Inc.'s gpsOne™ solution terminals as subjects instead of GSM types or regular GPS devices.

2.19.1. EVENT

MH_GPS_EVENT must be additionally defined for location information HAL API. Likewise, the MH_GPSEvent structure must be transferred as the argument.

The platform receiving MH_GPS_EVENT from the terminal must report that the GPS-related result has arrived by transferring the event to the application where the GPS information was requested.

The platform may not allow GPS-related requests made on multiple applications, maintaining a single request instead.

An event that has occurred has the following values:

```
typedef enum MH_GPSSubEvent {
        MH_GPSEV_SUCCESS = 0x1,  // Successful GPS data reception
        MH_GPSEV_FAILED,         // GPS data reception failed
        MH_GPSEV_BEGIN,          // GPS starts without error as requested
        MH_GPSEV_NO_CON,         // Connection to PDE server failed
        MH_GPSEV_INPROGRESS,     // GPS device already in use
        MH_GPSEV_LOCKED          // GPS locked
} MH_GPSSubEvent;
```

The sub-event above is transferred through the following structure:

```
typedef struct MH_GPSEvent{
        MH_GPSSubEvent event;            // GPS event that occurs,
} MH_GPSEvent;
```

2.19.2. Global Structure

The following structure should be defined for location information HAL API:

```
typedef struct MH_gpsConfig {
        M_Uint8 mode;                   // Operation mode
        M_Uint8 qos;                    // Quality of location information
        M_Uint16 transport;             // Transport layer of location information
        M_Uint32 pde_addr;              // PDE server address
        M_Uint16 pde_port;              // PDE server port
} MH_gpsConfig;
typedef enum MH_gpsMode {
      MH_MS_ASSISTED = 0x1,         // MS_ASSISTED mode
        MH_MS_BASED,                // MS_BASED mode
        MH_OPT_SPEED,               // Speed optimization mode
        MH_OPT_ACCURACY             // Accuracy optimization mode
}
typedef enum MH_gpsTransport {
 SERVER_TCPIP=0x1,
     SERVER_DBURST=0x2;
}



typedef struct MH_locationInfo {
        M_Int32 latitude;       //Latitude
        M_Int32 longitude;      //Longitude
        M_Int16 altitude;       //Altitude
        M_Uint16 heading;       //Heading of location
        M_Uint16 velocityHor;   //Horizontal velocity
        M_Int8 velocityVer;     //Vertical velocity
        M_Int8 accuracy;        //Data accuracy
        M_Char* timeString;     //Time string(hhmmss)
} MH_locationInfo;
typedef enum MH_gpsCfgMask {
        MH_GPSCFG_OPTI=0x1,     // gpsOne™ operation mode applied
        MH_GPSCFG_QOS=0x2,      // gpsOne™ quality standard applied
```

```
        MH_GPSCFG_TRANSPORT=0x4,        // gpsOne™ data transmission layer
                                        applied
        MH_GPSCFG_SVRADDR=0x8,          // PDE server address value
                                        applied
        MH_GPSCFG_SVRPORT=0x10          // PDE connection port value
                                        applied
        MH_GPSCFG_ALL=0xFFFF,           // Composition information setup
} MH_gpsCfgMask;
```

- **MH_gpsAvailable**

M_Int32 MH_gpsAvailable(void)

Description

API that determines whether the gpsOne device exists

Parameters

None

Return Value

Pass

0 Device available

Fail

M_E_ERROR Device unavailable

Side Effects

None

Reference Item

None

- **MH_gpsRequestLocationInfo**

M_Int32 MH_gpsRequestLocationInfo(M_Int32 interval)

Description

This is API requesting for location info through gpsOne and operating asynchonically.
The result is reported as an event, with detailed info obtainable from MH_gpsGetResult().

Parameters

[in]	interval	Transmission termination at -1; at 0, reporting every second when greater than 1 (if possible)

Return Value

Pass

0

Fail

M_E_INVALID	Invalid parameter
M_E_INUSE	Currently in use
M_E_ERROR	Other errors

Side Effects

None

Reference Item

None

- **MH_gpsGetResult**

M_Int32 MH_gpsGetResult(MH_locationInfo pInfo)*

Description

This function is called when the subevent of the GPS event is MH_GPSEV_SUCCESS and should be returned with the result of the location information in pInfo.

Parameters

[out] pInfo Location info obtained through the MH_locationInfo structure

Return Value

Pass

0

Fail

M_E_ERROR Other errors

Side Effects

None

Reference Item

None

- **MH_gpsGetConfig**

*M_Int32 MH_gpsGetConfig(MH_gpsConfig *config)*

Description

Obtains the gpsOne™ setup data

Parameters

[out] config Structure's pointer for receiving the terminal's gpsOne™ setup data

Return Value

Pass

0 Data acquisition success

Fail

M_E_ERROR Data acquisition failure

Side Effects

None

Reference Item

None

- **MH_gpsSetConfig**

*M_Int32 MH_gpsSetConfig(MH_gpsConfig *config, MH_gpsCfgMask mask)*

Description

Sets up the gpsOne™ information

Parameters

[in]	config	Structure's pointer including composition data to be set up
[in]	mask	Application subject category mask from the structure contents

Return Value

Pass

0	Setup success

Fail

M_E_ERROR	Setup failure
M_E_INVALID	When the set data is invalid

Side Effects

None

Reference Item

None

- **MH_gpsControl**

M_Int32 MH_gpsControl(M_Int32 function, M_Int32 command, void argument)*

Description

API prepared for later expansion; no function has been defined in the current version of standards, thereby returning M_E_ERROR

Parameters

[in]	function	Expansion feature type
[in]	command	Expansion feature command type
[in]	argument	Pointer of the factor's structure on the command

Return Value

Pass

0

Fail

M_E_ERROR

Side Effects

None

Reference Item

None