AMOITECH

Mercury API SPEC

Rev. 2.3.3
2019/7/19

Revision History:

Date	Version	Describe	Author	Reviewer
2017.02.21	V1.0	first draft	Siquan.huang	
2017.03.03	V1.1	Update com/spi/nv/keypad/sms apis	Siquan.huang	
2017.03.21	V1.2	Add socket api , update file system api, some apis parameters and return value revise.	Siquan.huang	
2017.04.14	V1.3	Add DNS parsing api, update graphics api, add public api. delete midi api	Siquan.huang	
2017.04.20	V1.4	Update tts api, update tp/barscan api.	Siquan.huang	
2017.04.25	V1.4.1	Add flight mode api, gprs attach and detach in net api. add SetLocalTime and GetLocalTime in timer api.	Siquan.huang	
2017.04.25	V1.5.0	Update audio api. add DisplayGetDirection, DisplaySetDirection, DisplayScreenOn in graphics, thread/syn/memory/timer/public/uart/spi/nv/telephony/net/audio return value modify	Siquan.huang	
2017.05.05	V1.5.1	Add poweroff/powerreboot in power management api; tts/filesystem return value modify; add RegNotifyCallback in public	Siquan.huang	
2017.05.09	V1.5.2	Add NV_AppInfoRead/ NV_AppInfoWrite in nv, nled id only support led-1. remove touch api	Siquan.huang	
2017.05.12	V1.5.3	Add Cam_QR_Enc in barscan, change Cam_StartScan that scan results via receive message from noitifycallback. add DisplayTransparentString in graphics.	Siquan.huang	
2017.05.19	V1.6.0	Add AudioDtmfPlay, AudioDtmfAbort in audio api. add rgb color index table, add DisplayRLE_BMP, DisplayPaintEnd in graphics. update Cam_StartScank; add Power_GetVoltagetoPercent in power mangement, delete Battery(charge); remove tts TTS_GetParams TTS_SetParams api	Siquan.huang	

2017.05.26	V1.6.1	Add com baud_rate values macro: Com Baud Rate; remove all message about tp; Tidy up Type definition; bar scan qr encode add note, startscan mode parameter range value is 0 or 1. tts playtext parameters change; change SimGetInfo->GetSimInfo in platform public; fixed up GetCurrentTime explain wrong.	Siquan.huang	
2017.06.09	V1.6.2	Add notice catalog: explain unsigned data(-1 == 0xFFFFFFFF); NLed return value modify(use last error value); Filesystem add three api: FomatDisk/ GetDeviceFreeSpace / GetDeviceUsedSpace; Add FOTA api; add app entry api: Mercury_Main	Siquan.huang	
2017.06.15	V1.7.0	RGB565 index list update; Add InitFileSystem, UnInitFileSystem api in file system; add c std lib explain in Notes;	Siquan.huang	
2017.06.23	V1.7.1	Cam_StartScan add note; BARSCAN_MODE_VALUE_E add motion detect mode. add BatteryGetStatus api; add sim notify enumerations and tts notify enumerations.	Siquan.huang	
2017.06.29	V1.7.2	SocketConnect: the message reported data is socket id and error code; add ps on and off notify; Add network api: SocketGetOpt/SocketGetState Network_GetPDPStatus;	Siquan.huang	
2017.07.07	V1.7.3	NV_AppInfoRead/ NV_AppInfowrite len parameter is set to 640. SetThreadPriority can set thread priority from 100 to 255.	Siquan.huang	
2017.07.14	V1.7.4	Add PWM and I2C api; realize wakelock/wakeunlock interface;	Siquan.huang	
2017.07.28	V1.7.5	Update BatteryGetStatus: add USB charge state; in screen lock mode, short press power key, that will report short press notify.	Siquan.huang	

2017.08.04	V1.7.6	Barscan add two modes: BARSCAN_MODE_ONED_ONLY and BARSCAN_MODE_MOTIONDETECT_MINI	Siquan.huang	
2017.08.11	V1.7.7	GetMercuryVersion: Return content change	Siquan.huang	
2017.08.18	V1.7.9	1. Add System introduction 2. add notify id introduction 3. delete enumeration: MC_SMS_ERROR_E 4. BatteryGetStatus add: judge battery is exist.	Siquan.huang	
2017.08.25	V1.7.10	1. note: press power key only to light screen 2. graphics add api: DisplayBitMapGet/DisplayLineRGB565/LCD_SetColorRGB565. 3. fota add: APP_FlashWrite/ APP_FlashErase/ APP_FlashRead.	Siquan.huang	
2017.09.01	V1.7.11	1. add instruction for scan once mode.	Siquan.huang	
2017.09.07	V1.7.12	1. COM_CONFIG_T add explain 2.fix up: NV_Write is 0--499 3. com baud rate no support 3250000 4. add enum PDP_ID_E.	Siquan.huang	
2017.09.22	V1.8.2	1.add explain: file system only supports E disk 2.add four function for usb api in UART 3.add get base station information API in network 4.add RenameFile api	Siquan.huang	
2017.9.30	V1.8.3	1. add ADC_GetResult api 2.add DisplayMutilInit api 3.add LocalReAlloc api 4.add usb plug in/out notify msg	Siquan.huang	
2017.10.20	V1.8.4	1. correntions notes: FOTA_WroteLenGet 2. add api in fileSystem: SetFileSize and GetFilePointer 3.add spi dma switch api 4. add Network_NetAddrGet for	Siquan.huang	

		get ip address.		
2017.10.31	V1.8.5	1. update filesystem api: InitFileSystem, 2.add api in thread: MercurySendMessage/ MercuryGetMessage/ MercuryPeekMessage 3.add MercuryReadID in platform	Siquan.huang	
2017.11.08	V1.8.6	1. add dtmf play complete notify: MC_AUDIO_NOTIFY_ID_E 2. add marco definition for COM_CONFIG_T parameters. 3.Cam_StartScan add important note (20171109)	Siquan.huang	
2017.11.21	V1.8.7	1. update API: SmartCardReset, add reset Voltage.	Siquan.huang	
2017.12.14	V1.8.8	1. add api: Cam_Suspend 2. add api: DisplayPointRGB565 in real time	Siquan.huang	
2017.12.28	V1.8.10	1. update: add two states for fastboot states.(MercuryFastbootStateGet) 2. add Cam_Suspend	Siquan.huang	
2018.01.17	V1.8.11	1. add explain in symbol_type_t enum 2. add explain in filesystem for calling **FlushFileBuffers**	Siquan.huang	
2018.03.05	V1.8.12	1. Reorganize the API interface	Siquan.huang	
2018.03.16	V1.8.14	1. add audio api to set dtmf volume: AudioDtmfVolume	Siquan.huang	
2018.03.29	V1.8.15	1.add single tone api: AudioSingleTonePlay/ AudioSingleToneAbort/ AudioSingleToneVolume 2. barscan api: Cam_FeatureConfig 3. InitFileSystem: support to 16M	Siquan.huang	
2018.05.03	V1.9.0	1. add TTS api: TTS_SetParams / TTS_GetParams 2. add api: InitFileSystemPlus 3. add SSL API: mercury_ssl_establish\ mercury_ssl_destroy\	Siquan.huang	

		mercury_ssl_write\ mercury_ssl_read\ 4. gprs attach/deattach add notify message.		
2018.05.10	V1.9.2	1. add explain InitFileSystem / InitFileSystemPlus		
2018.08.14	V2.0.0	1. add api: KP_SetBlackLight to set keypad blacklight.	Siquan.huang	
2018.10.10	V2.0.3	1. add api in net: Network_ForceCampon\ Network_CancelForceCampon\ Network_SetGprsMassRetransmitParam 2. add api in power manager: SetChgOverHighTemp\ SetChgOverLowTemp\ SetRechgVol 3. add api in tts: PCM_StartPlay\ PCM_FillData\ PCM_StopPlay	Siquan.huang	
2018.10.30	V2.0.4	1. add api in Net: Network_SetAuthType\ Network_SelectBand\Network_PingRequest\Network_PingCancel\Network_GetTaPwr	Siquan.huang	
2018.12.19	V2.2.0	1.add api DisplayGetRGB565 in graphics 2.add api Cam_QR_SetEncLEVEL in bar Scan. 3. add Littlels interfaces. 4. Abandoning the SSL interfaces 5. 1)add interfaces for network: a. Network_PingV4Request b. Network_PingV6Request c. NetworkSetPdpType 2)deprecated Network_PingV4Request interface. 6. add interface in platform: MercuryLogoUpdata 7. add SHA1/SHA256 interfaces	Siquan.huang	
2019.1.30	V2.2.1	1. add about IPv6 api and other link bearer api in Net.(15 new api:36~50).	Siquan.huang	

		2. add api in TTS: a.AMR_StartPlay b. AMR_StopPlay 3. add api in platform: GetSdkVersion 4. add api in Fota: a. FOTA_RawDataInfoSet b. FOTA_RawDataClear		
2019.03.28	V2.2.2	1.add api in poower manager: a. SetChgEndVol b. SetChgSwtich c. SetAutoIdentAdp d. GetAdpType 2. add api in fota: a. LOGO_FlashWrite b. LOGO_FlashErase c. LOGO_FlashRead	Siquan.huang	
2019.07.04	V2.3.0	1.add api in net: a) SocketCreateV6 b) SocketBind c) SocketListen d) SocketAccept e) SocketSelect f) FdClr g) FdSet h) FdIsSet i) FdZero 2. add api in platefrom: MercuryGetPsRwMem 3.add api in bar scan: a) Cam_QR_SetScanDensity b) Cam_QR_GetScanDensity c) Cam_CfgSymbolEnable d) Cam_GetBarLibVersion e) Cam_SetPrescanLine	Siquan.huang	
2019.07.19	V2.3.3	1.add api in lfs: a) FileSysUseCap b) LittlefsVersion	Siquan.huang	
2019.08.06	V2.3.4	1. update enum: BARSCAN_MODE_VALUE_E	Siquan.huang	

Mercury APIs

Catalog

Notes:

1. If the unsigned data type is equals to -1, then -1 is equivalent to 0xff. such as , uint32 a equals to -1, then a equals to 0xffffffff also.

2. For the C standard library support, the user should include the "MercuryCSTD.h" header file where it is needed. And don't add the other C standard library files again.

3. In sleep state, it just can press the power key only to light the screen.

System introduction

This system is a embedded real-time operating system. The system supports priority preemption, no time slice, and the priority levels value can range from zero through 255, with zero as the highest priority.

Critical thread description:

The priority of LCD maintenance thread is 150.
The priority of scan code processing thread is 38.
The priority of notify thread(RegNotifyCallback) is 200.
The priority of timer processing thread is 44.
The priority of GPIO interrupt processing thread is 44.
The priority of the Mercury_Main thread, which is Mercury Platform jump to the app entry function, is 44, and the stack size is 64k byte.

App can create and modify the threads with priority ranges from 100 to 255.

App design, please in conjunction with priority preemption mode features, rationally allocate the use of CPU resources.

Notice that the scan code needs to consume more CPU resources, and the higher priority, so please try not to do GPRS communication and refresh and so also needs a large number of CPU resources in action.

About scan code mode, please do not use the continuous scan mode directly unless you are under test. Or you can contact us, we will introduce the appropriate scan code mode.

App available system resources:

512kbyte heap + 512kbyte RW
1024kbyte flash

App entry

void Mercury_Main(uint32 bootMode)

App must implement Mercury_Main for platform calls.
bootMode:

0 ------ charge boot
1 ------ normal boot

Kernel Services

Thread

1、CreateThread

This function creates a thread to execute within the address space of the calling process.

Syntax

```
HANDLE CreateThread(
        LPSECURITY_ATTRIBUTES lpsa,
        DWORD cbStack,
        LPTHREAD_START_ROUTINE lpStartAddr,
        ULONG argc,
        LPVOID lpvThreadParam,
        DWORD fdwCreate,
        LPDWORD lpIDThread
);
```

Parameters

lpsa

[in] Ignored. Set to NULL. See LPSECURITY_ATTRIBUTES.

cbStack

[in] Ignored unless the STACK_SIZE_RESERVATION flag is used. In that case, this parameter specifies the virtual memory reserved for the new thread. There is no limit to the size of the stack distribution theory, but it can't be larger than the remainder of the total distribution. The platform has 512k bytes heaps for APP, but this part is used intersecting with the platform, so the APP stack is best used not to exceed the 300K bytes.

If this parameter is zero, the new thread uses the default size for the executable.. The default stack size is 8k bytes.

lpStartAddr

[in] Long pointer to the application-defined function of type LPTHREAD_START_ROUTINE to be executed by the thread; represents the starting address of the thread. See LPTHREAD_START_ROUTINE

argc

[in] The first parameter of the thread entity function.

lpvThreadParam

[in] Long pointer to a single 32-bit parameter value passed to the thread.

fdwCreate

[in] Specifies flags that control the creation of the thread.

The following table shows the values for this parameter.

Value	Description
CREATE_SUSPENDED	4, The thread is created in a suspended state and does not run until the **ResumeThread** function is called. The cbStack parameter is invalid.
STACK_SIZE_RESERVATION	0x1000, The cbStack parameter specified the maximum stack size instead of being ignored.

lpIDThread

[out] Long pointer to a 32-bit variable that receives the thread identifier. If this parameter is NULL, the thread identifier is not returned.

Return Value

A handle to the new thread indicates success. NULL indicates failure. To get extended error information, call GetLastError.

2、ExitThread

This function ends a thread.

Syntax

```
VOID ExitThread(
    DWORD dwExitCode
);
```

Parameters

dwExitCode

[in] Specifies the exit code for the calling thread. Here set to NULL.

Return Value

None

3、GetCurrentThread

This function returns a pseudo handle for the current thread. A pseudo handle is a special constant that is interpreted as the current thread handle.

Syntax

```
HANDLE GetCurrentThread(void);
```

Return Value

Pseudo handle for the current thread.

4、GetCurrentThreadId

This function returns the thread identifier, which is used as a handle of the calling thread

Syntax

```
DWORD GetCurrentThreadId(void);
```

Parameters

None

Return Value

The thread identifier of the calling thread indicates success.

5、GetThreadPriority

This function returns the priority value for the specified thread

Syntax

```
int GetThreadPriority(
        HANDLE hThread
);
```

Parameters

hThread

[in] Handle to the thread.

Return Value

If success, return the thread priority level. -1 indicates failure。

6、ResumeThread

This function is called to resume execution of a thread that was suspended by the SuspendThread member function, or a thread created with the **CREATE_SUSPENDED** flag.

Syntax

```
DWORD ResumeThread(
    HANDLE hThread
);
```

Parameters

hThread

[in] Specifies a handle for the thread to be restarted. The specified thread must be suspended before.

Return Value

Zero indicates success. 0xFFFFFFFF indicates restart failure. 0x12 indicates specified thread is not suspended.

7、SetThreadPriority

This function sets the priority value for the specified thread.

Syntax

```
BOOL SetThreadPriority(
    HANDLE hThread,
```

```
    int nPriority
);
```

Parameters

hThread

[in] Handle to the thread whose priority value is to be set.

nPriority

[in] Specifies the priority value for the thread. The possible range of priority is from 100 to 255. if the input value is less than 100, automatically set to 100.

Return Value

Return zero always, user can call **GetThreadPriority** to see if the setting are successful.

8、Sleep

This function suspends the execution of the current thread for a specified interval.

Syntax

```
void Sleep(
    DWORD dwMilliseconds
);
```

Parameters

dwMilliseconds

[in] Specifies the time, in milliseconds, for which to suspend execution. A value of zero causes the thread to relinquish the remainder of its time slice to any other thread of equal priority that is ready to run. If no other threads of equal priority are ready to run, the function returns immediately, and the thread continues execution. A value of INFINITE causes an infinite delay. The value of INFINITE is -1 (0xFFFFFFFF).

Return Value

None

9、SuspendThread

This function suspends the specified thread

Syntax

```
DWORD SuspendThread(
    HANDLE hThread
);
```

Parameters

hThread

[in] Handle to the thread.

Return Value

Zero indicates success. 0xFFFFFFFF indicates restart failure.

10、TerminateThread

This function stops the specified thread

Syntax

```
BOOL TerminateThread(
    HANDLE hThread,
    DWORD dwExitCode
);
```

Parameters

hThread

[in] Handle to the thread to terminate

dwExitCode

[in] Reserved, Set to NULL.

Specifies the exit code for the thread. To retrieve a thread's exit value, use the GetExitCodeThread function.

Return Value

TRUE indicates success. FALSE indicates failure.

11、MercurySendMessage

This function sends the message from the current thread to the specified thread.

Syntax

```
int MercurySendMessage(
                        MERCURY_MESSAGE_S* msg,
                        HANDLE threadID
                    );
```

Parameters

msg

[in]The message that wants to send. The msg can't equal to NULL. See MERCURY_MESSAGE_S.

threadID

[in]The Thread ID for receiving messages. The thread ID must be valid.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError

12、MercuryGetMessage

This function gets the message sent to the thread of ID is threadID. This function is a block function, if threadID does not have message, hang the current thread until threadID receives message. If threadID has message, the function returns immediately.

Syntax

```
int MercuryGetMessage(
                        MERCURY_MESSAGE_S** msg,
                        HANDLE threadID
                    );
```

Parameters

msg

[out]Pointer to the receive message. See MERCURY_MESSAGE_S.

threadID

[in]The thread ID, specify which thread to receive.

Return Value

Return zero always.

13、MercuryPeekMessage

This function gets the message sent to the thread of ID is threadID. This function is a non-block function, if threadID does not have message, After calling the function, no matter whether the threadID has message or not, it is returned immediately.

Syntax

```
int MercuryPeekMessage(MERCURY_MESSAGE_S** msg, HANDLE threadID);
```

Parameters

msg

[out] Pointer to the receive message. See MERCURY_MESSAGE_S.

threadID

[in] The thread ID, specify which thread to receive.

Return Value

Zero indicates receive message success. -1 indicates the threadID does not have message.

Synchronization

This reference section contains descriptions of kernel synchronization programming elements.

1、 CreateEvent

This function creates a named or an unnamed event object.

Syntax

```
HANDLE CreateEvent(
    LPSECURITY_ATTRIBUTES   lpEventAttributes,
    BOOL bManualReset,
    BOOL InitialState,
    LPTSTR lpName
);
```

Parameters

lpEventAttributes

[in] Ignored. Set to NULL. See LPSECURITY_ATTRIBUTES

bManualReset

[in] Boolean that specifies whether a manual-reset or auto-reset event object is created. If TRUE, then you must use the ResetEvent function to manually reset the state to nonsignaled. If FALSE, the system automatically resets the state to nonsignaled after a single waiting thread has been released.

InitialState

[in] Boolean that specifies the initial state of the event object. If TRUE, the initial state is signaled; otherwise, it is nonsignaled.

lpName

[in] Pointer to a null-terminated string that specifies the name of the event object. The name is limited to MAX_PATH characters and can contain any character except the backslash path-separator character (\). Name comparison is case sensitive.

If *lpName* matches the name of an existing named event object, the *bManualReset* and *bInitialState* parameters are ignored because they have already been set by the creation process.

If *lpName* is NULL, the event object is created without a name.

Return Value

A handle to the event object indicates success. If the named event object existed before you call this function, **CreateEvent** returns a handle to the existing object . -1 indicates failure. To get extended error information, call GetLastError.

2、 DeleteCriticalSection

This function releases all resources used by a critical section object that is not owned.

Syntax

```
void DeleteCriticalSection(
    LPCRITICAL_SECTION lpCriticalSection
);
```

Parameters

lpCriticalSection

[in] Pointer to the critical section object. See LPCRITICAL_SECTION.

Return Value

None. To get extended error information, call GetLastError.

3、 EnterCriticalSection

This function waits for ownership of the specified critical section object. The function returns when the calling thread is granted ownership.

Syntax

```
void EnterCriticalSection(
    LPCRITICAL_SECTION lpCriticalSection
);
```

Parameters

lpCriticalSection

[in] Pointer to the critical section object. See LPCRITICAL_SECTION

Return Value

None. To get extended error information, call GetLastError.

4、 InitializeCriticalSection

This function creates and initializes a critical section object.

Syntax

```
void InitializeCriticalSection(
    LPCRITICAL_SECTION lpCriticalSection
);
```

Parameters

lpCriticalSection

[in] Pointer to the critical section object, the parameter can't be NULL. See LPCRITICAL_SECTION

Return Value

None. To get extended error information, call GetLastError.

5、 LeaveCriticalSection

This function releases ownership of the specified critical section object.

Syntax

```
void    LeaveCriticalSection(
        LPCRITICAL_SECTION lpCriticalSection
);
```

Parameters

lpCriticalSection

[in] Pointer to the critical section object. See LPCRITICAL_SECTION

Return Value

None. To get extended error information, call GetLastError.

6、 OpenEvent

This function opens an existing named event object.

Syntax

```
HANDLE OpenEvent(
    DWORD dwDesiredAccess,
    BOOL bInheritHandle,
    LPCTSTR lpName
);
```

Parameters

dwDesiredAccess

[in] Specifies the requested access to the event object. For systems that support object security, the function fails if the security descriptor of the specified object does not permit the requested access for the calling process. Reserved, can be set to 0.

bInheritHandle

[in] Specifies whether the returned handle is inheritable.

Reserved ,set to FALSE.

lpName

[in] Pointer to a null-terminated string that names the event to be opened. Name comparisons are case-sensitive. Each object type, such as memory maps, semaphores, events, message queues, mutexes, and watchdog timers, has its own separate namespace. Empty strings, "", are handled as named objects. On Windows desktop-based platforms, synchronization objects all share the same namespace.

Return Value

A handle to the event object indicates success. -1 indicates failure. To get extended error information, call GetLastError.

7、 PulseEvent

This function provides a single operation that sets to signaled the state of the specified event object and then resets it to nonsignaled after releasing the appropriate number of waiting threads.

Syntax

```
BOOL PulseEvent(
    HANDLE hEvent
);
```

Parameters

hEvent

[in] Handle to the event object. The **CreateEvent** function returns this handle.

Return Value

Zero indicates success. 1 indicates failure. To get extended error information, call GetLastError.

8、 ResetEvent

This function sets the state of the specified event object to nonsignaled.

Syntax

```
BOOL ResetEvent(
    HANDLE hEvent
);
```

Parameters

hEvent

[in] Handle to the event object, returned by the CreateEvent function.

Return Value

Zero indicates success. 1 indicates failure. To get extended error information, call GetLastError.

9、 SetEvent

This function sets the state of the specified event object to signaled.

Syntax

```
BOOL SetEvent(
    HANDLE hEvent
```

```
);
```

Parameters

hEvent

[in] Handle to the event object, returned by the CreateEvent function.

Return Value

Zero indicates success. 1 indicates failure. To get extended error information, call GetLastError.

10、 WaitForSingleObject

This function returns when the specified object is in the signaled state or when the time-out interval elapses.

If there have multiple threads, wait for an event at the same time. When this event is set, all the threads can get the single.

Syntax

```
DWORD WaitForSingleObject(
  HANDLE hHandle,
  DWORD dwMilliseconds
);
```

Parameters

hHandle

[in] Handle to the object. For a list of the object types whose handles can be specified, see the Remarks section.

dwMilliseconds

[in] Specifies the time-out interval, in milliseconds. The function returns if the interval elapses, even if the object's state is nonsignaled. If dwMilliseconds is zero, the function tests the object's state and returns immediately. If dwMilliseconds is INFINITE(0xffffffff), the function's time-out interval never elapses.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

Memory

This reference section contains descriptions of memory management programming elements.

1、 LocalAlloc

This function allocates the specified number of bytes from the heap.

Syntax

```
HLOCAL LocalAlloc(
    UINT uFlags,
    UINT uBytes
);
```

Parameters

uFlags

[in] Specifies how to allocate memory. If zero is specified, the default is the LMEM_FIXED flag. The following table shows possible values.

Value	Description
LMEM_FIXED	Allocates fixed memory. The return value is a pointer to the memory object.
LMEM_ZEROINIT	Initializes memory contents to zero.
LPTR	Combines the LMEM_FIXED and LMEM_ZEROINIT flags.

Currently, this parameter can be ignored and you can set it to 0.

uBytes

[in] Specifies the number of bytes to allocate, unit is byte.

Return Value

A handle to the newly allocated memory object indicates success. NULL indicates failure. To get extended error information, call GetLastError.

2、 LocalReAlloc

This function changes the size or the attributes of a specified local memory object. The size can increase or decrease.

Syntax

HLOCAL LocalReAlloc(void * memblock,UINT size);

Parameters

memblock

[in] Handle to the local memory object to be reallocated.

This handle is returned by either the **LocalAlloc** or the **LocalReAlloc** function.

size

[in] New size, in bytes, of the memory block.

Return Value

A handle to the reallocated memory object indicates success. NULL indicates failure. To get extended error information, call GetLastError.

3、 LocalFree

This function frees the specified local memory object and invalidates its handle.

Syntax

```
HLOCAL LocalFree(
    HLOCAL hMem
);
```

Parameters

hMem

Handle to the local memory object. This handle is returned by either the LocalAlloc or LocalReAlloc function.

Return Value

NULL indicates success. A handle to the local memory object indicates failure. To get extended error information, call GetLastError.

4、 SecureZeroMemory

This function fills a block of memory with zeros.

Syntax

```
LPVOID SecureZeroMemory(
        LPVOID ptr,
        SIZE_T cnt
);
```

Parameters

ptr

[in] Pointer to the starting address of the block of memory to fill with zeros.

cnt

[in] Size, in bytes, of the block of memory to fill with zeros.

Return Value

A pointer to the block of memory. NULL indicates failure. To get extended error information, call GetLastError.

Timer

This reference section contains descriptions of time programming elements.

1、 CreateTimer

The function create a timer with call back function. A maximum of 50 timers can be created at the same time.

Aperiodic timer, when the timer reaches, call callback function timer_fun, The timer stops running, but it still exists, until call the DeleteTimer to delete it.Periodic timer, when the timer reaches, call the callback function timer_fun, and the timer is restarted.

Syntax

```
TIMER_PTR CreateTimer(TIMER_CONFIG_S* cfg);
```

Parameters

cfg

[in]The structure to initialize timer's parameters. See TIMER_CONFIG_S.

Return Value

A pointer to the control block of the timer indicates success, See TIMER_PTR. NULL indicates failure. To get extended error information, call GetLastError.

2、 ChangeTimer

The function changed timer's callback function and expire time. User have to call DeactiveTimer function to pause timer before change timer , and then have to call the ActiveTimer after change timer for reactive the timer.

Syntax

```
INT ChangeTimer(
    TIMER_PTR timer_ptr,
    TIMER_FUN timer_fun,
    ULONG timer_expire
);
```

Parameters

timer_ptr

[in]Pointer to a timer that has been created and to be changed.

timer_fun

[in]Timer callback function, it can't be set to NULL.

timer_expire

[in] Specifies the expire value in milliseconds, it can't be equal to 0.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

3、 ActiveTimer

The function activates a timer created before.

Syntax

INT ActiveTimer(TIMER_PTR timer_ptr);

Parameters

timer_ptr

[in]Pointer to a previously created application timer.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

4、 DeactiveTimer

The function deactivates a timer created before. When the timer is paused, the remaining time is saved, and when it starts again, the timer starts from the rest of the time.

Syntax

INT DeactiveTimer(TIMER_PTR timer_ptr);

Parameters

timer_ptr

[in]Pointer to a previously created application timer.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

5、 IsTimerActive

The function checks that the timer is still active.

Syntax

INT IsTimerActive(TIMER_PTR timer_ptr);

Parameters

timer_ptr

[in]Pointer to a previously created application timer.

Return Value

If it is active, returns 1, and 0 indicates not active. -1 indicates failure. To get extended error information, call GetLastError.

6、 DeleteTimer

The function deletes a timer created before.

Syntax

INT DeleteTimer(TIMER_PTR timer_ptr);

Parameters

timer_ptr

[in]Pointer to a previously created application timer.

Return Value

Zero indicates success, the pointer to timer must set to null after deleted. -1 indicates failure. To get extended error information, call GetLastError.

7、 GetCurrentTime

This function retrieves the number of seconds that have elapsed since the system was started. Its unit is second.

Syntax

INT GetCurrentTime(ULONG* time);

Parameters

time
[out] Pointer to the number of seconds that have elapsed since the system was started.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

8、 GetTickCount

This function retrieves the number of Ticks that have elapsed since the system was started. At present, 1 tick equals 1ms.

Syntax

INT GetTickCount(ULONG* tick);

Parameters

tick
[out] Pointer to the number of Ticks that have elapsed since the system was started.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

9、 SetLocalTime

This function sets the current local time and date(year/month/day of week/day/hour/minute/second).

Syntax

int SetLocalTime(const SYSTEMTIME* lpSystemTime);

Parameters

lpSystemTime
[in] Pointer to a SYSTEMTIME structure that contains the current local date and time. The **wDayOfWeek** member of the SYSTEMTIME structure is ignored.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

The following table shows the possible error values.

Value	Description
ERROR_TM_WRONG_DATE	0x800c0002, the date is out of range(year/month/day of week/day).
ERROR_TM_WRONG_TIME	0x800c0003, the time is out of range(hour/minute/second).

10、 GetLocalTime

This function retrieves the current local date and time (year/month/day of week/day/hour/minute/second/milliseconds).

Syntax

void GetLocalTime(SYSTEMTIME* lpSystemTime);

Parameters

lpSystemTime

[out] Pointer to a SYSTEMTIME structure to receive the current local date and time.

Return Value

None.

Platform public api

1. GetSimInfo

This function get the SIM card information, contains IMSI, CCID.

Syntax

```
int GetSimInfo(
    SIM_INFO_E flag,
    SIM_IMSI_T* imsi_str,
    SIM_ICCID_T* ccid
    );
```

Parameters

flag
[in]Set which info want to get. See SIM_INFO_E.

imsi_str
[out]Pointer to the structure that about IMSI info. See SIM_IMSI_T.

ccid
[out]Pointer to the structure that about CCID info. See SIM_ICCID_T.

Return Value

1 indicates success. Zero indicates failure, it means that SIM card isn't ready. The status of SIM is reported by message notification.

2. GetImei

This function get the device IMEI number.

Syntax

```
BYTE* GetImei(void);
```

Parameters

None.

Return Value

Pointer to a string containing the IMEI information indicates success. NULL indicates failure.

3. GetMercuryVersion

The function gets the current version number of the platform.

Syntax

BYTE* GetMercuryVersion(void);

Parameters

None.

Return Value

If successful, returns the platform version number and manufacturer information. Null indicates failure.

4. MercuryDebug

The function print out the debug log data. The data output from SPRDU2S Diag port.

Syntax

```
void MercuryDebug(
    uint8 *buffer,
    uint32 length
    );
```

Parameters

buffer
[in] Pointer to the buffer block for the debugger.

length
[in]The size of the debug data block that will to be written.

Return Value

None.

5. GetLastError

This function returns the calling thread's last-error code value. You should call the **GetLastError** function immediately when a function's return value indicates that such a call will return useful data. That is because some functions call **SetLastError**(0) when they succeed, wiping out the error code set by the most recently failed function.

Syntax

DWORD GetLastError(void)

Parameters

None.

Return Value

The calling thread's last-error code value indicates success. For a complete list of error codes, see the SDK header file **MercuryErrorValue.h** .

6. RegNotifyCallback

This function registers a NotifyCallback function to deal with message from platform.

Syntax

void RegNotifyCallback(NotifyCallback fun);

Parameters

fun

[in] A NotifyCallback function that will be called back when a notification arrives. For a prototype for this function, see NotifyCallback.

Return Value

None.

7. MercuryReadID

This function gets the module unique identifier. The length is 9 bytes.

Syntax

BOOLEAN MercuryReadID(uint8* MercuryID);

Parameters

MercuryID

[out]Pointer to the module unique identifier, and the length is 9 bytes.

Return Value

TURE indicates success. FALSE indicates failure.

8. MercuryLogoUpdata

This function allows user to update logo in APP. Logo data have to be stored in code or file system first.

Syntax

```
int MercuryLogoUpdata(
        MERCURY_BOOT_IMAGE_S                    *bootImageInfo,
        MECURY_LOGO_UPDATA_CALLBACK_FPTR callback_fptr,
        uint32 len
    );
```

Parameters

bootImageInfo
[in]The logo property. See MERCURY_BOOT_IMAGE_S.
callback_fptr
[in]The callback function to load logo data. See MECURY_LOGO_UPDATA_CALLBACK_FPTR.
len
[in]The parameter of MECURY_LOGO_UPDATA_CALLBACK_FPTR.

Return Value

Zero indicates success. others indicates failure. To get extended error information, call GetLastError.

The following table shows possible values.

Value	Description
ERROR_LOGO_UPDATA_NONSUPPORT	The version is not support the interface.
ERROR_LOGO_UPDATA_BOOTIMAGEINFO	The **bootImageInfo** is NULL
ERROR_LOGO_UPDATA_MAGICNUM	Magic number error, magic number have to: 0x5a5aa5a5
ERROR_LOGO_UPDATA_IMAGELEN	The image is too big. it cann't greater to 150K
ERROR_LOGO_UPDATA_CALLBACK	The **callback_fptr** is NULL.
ERROR_LOGO_UPDATA_LEN	The **len** equals to zero.
ERROR_LOGO_UPDATA_MALLOC	Malloc buffer fail.
ERROR_LOGO_UPDATA_COPPY	Load logo map data failure.

9. GetSdkVersion

This function gets the SDK version.

Syntax

char* GetSdkVersion(void)

Parameters

None.

Return Value

Return the SDK version.

10. MercuryEIDGet

The function get the ESIM ID.

The result is reported by a notification message. The notification class is **NOTIFY_CLASS_STK**, and notification id is MC_STK_NOTIFY_ID_E. The data size is 8 bytes reported by the message, and the first 4 bytes are ESIM id, the last 4 bytes are the length of the ESIM ID.

Syntax

int MercuryEIDGet(void);

Parameters

None.

Return Value

0 indicates success, other indicates failure.

11. MercuryGetPsRwMem

This function gets the internal RW memory of PS.

Syntax

uint8* MercuryGetPsRwMem(int type, int *pBufSize);

Parameters

type
[in]Unuse, Reserve.
pBufSize
[out] the internal RW memory size of PS.

Return Value

Retrun a pointer to the internal RW memory of PS.

Graphics

1. DisplayInit

This function initializes the LCD. The initialization function must have already called before All the other functions are called. The DisplaySetColor function must be called after initialization, because the default values of background and foreground colors are black.

Syntax

int DisplayInit(void);

Parameters

None.

Return Value

Equal zero indicates success.

2. DisplayMutilInit

This function initializes the LCD by different LCD type. The initialization function must have already called before All the other functions are called. The DisplaySetColor function must be called after initialization, because the default values of background and foreground colors are black.

Syntax

int DisplayMutilInit(LCD_TYPE lcdType);

Parameters

lcdType

[in]The LCD type. The following table shows the possible values.

LCD Type	Description
ST7789H2_3WIRE_9BIT_2DATA	0
ST7789H2_4WIRE_8BIT_1DATA	1
ST7789V2_3WIRE_9BIT_2DATA	2
ILI9342C_4WIRE_8BIT_1DATA	3
ST7735_4WIRE_8BIT_1DATA	4

Return Value

Zero indicates success. -1 indicate failure.

3. DisplayBitMap

This function display the bitmap.

Syntax

```
int DisplayBitMap(
    RECTL* prclTrg,
    const LPBYTE *ImageData
    );
```

Parameters

prclTrg

[in] Pointer to a **RECTL** structure that defines the area to be modified, the unit of coordinate is Pixel, the coordinate value starts at 0, and a pixel contains three dot (RGB), with two bytes. The resolution of the device screen is QVGA(240X320).

ImageData

[in]Pointer to bitmap data. It is a uncompressed RGB565 array.

Return Value

Return zero, display bitmap in the specified area indicates success. -1 indicates failure.

4. DisplayBitMapGet

This function gets the bitmap data in the specified rectangular area.

Syntax

```
int DisplayBitMapGet(RECTL* prclTrg,    uint16 *ImageData, uint32 len);
```

Parameters

prclTrg

[in] Pointer to a **RECTL** structure that the data area you want to get. The prclTrg .width* prclTrg .height can't greater than **len**.

ImageData

[out]Pointer to the bitmap data that you get.

len

[in]The length of image data, in pixels, that you want to get.

Return Value

Equal zero indicates success. -1 indicate failure.

5. DisplayRLE_BMP

This function uses to display bitmap by BLE.

Syntax

```
int DisplayRLE_BMP(
    RECTL * prclTrg,
    const LPBYTE *ImageData,
    uint32 DataLen
);
```

Parameters

prclTrg

[in] Pointer to a **RECTL** structure that defines the area to be modified, the unit of coordinate is pixel, the coordinate value starts at 0, and a pixel contains three dot (RGB), with two bytes. The resolution of the device screen is QVGA(240X320).

ImageData

[in] Pointer to the bitmap data that wants to display. This data is a RGB565 array of lossless compression generated by the **BMP2RLE.exe** tool. The **BMP2RLE.exe** tool's storage path in SDK is ..\tools\BMP2RLE.

DataLen

[in]The size of the ImageData , in bytes, that will display. The parameter must be equal to the size of the **ImageData** bytes, otherwise the picture can't be displayed correctly.

Return Value

Zero indicates success. -1 indicates failure.

6. DisplayString

This function displays characters on the screen. SDK 1.8.4 and the above version, due to the removal of the built-in Chinese font library, so it can only display letters and Arabia numbers. The Chinese font library needs to be hung out.

Syntax

```
int DisplayString(
  POINT * prclTrg,
  const LPBYTE *String
```

```
);
```

Parameters

prclTrg

[in] Pointer to a POINT structure that defines the area to be displayed, the unit of coordinate is Pixel, the coordinate value starts at 0, and a pixel contains three dot (RGB), with two bytes. The resolution of the device screen is QVGA(240X320).

String

[in] String to be displayed on the screen.

Return Value

Zero indicates success, display string in the specified area. -1 indicates failure.

7. DisplayTransparentString

This function displays a string in the specified area with transparency. The difference between this function and the DisplayString is that the background of the string display is transparent.

SDK 1.8.4 and the above version, due to the removal of the built-in Chinese font library, so it can only display letters and Arabia numbers. The Chinese font library needs to be hung out.

Syntax

```
int DisplayTransparentString(
    POINT * prclTrg,
    const LPBYTE *String
);
```

Parameters

prclTrg

[in] Pointer to a POINT structure that defines the area to be displayed, the unit of coordinate is Pixel, the coordinate value starts at 0, and a pixel contains three dot (RGB), with two bytes. The resolution of the device screen is QVGA(240X320).

String

[in]String to be displayed on the screen.

Return Value

Zero indicates success, display string in the specified area. -1 indicates failure.

8. DisplayHorizLine

This function set horizontal display on the screen.

Syntax

```
int DisplayHorizLine(
    POINT * prclTrg,
    uint32 Length,
    uint32 Width,
    MERCURY_PALETEE_ARRAY_INDEX_E Color
    );
```

Parameters

prclTrg

[in] Pointer to a POINT structure that defines the area to be displayed, the unit of coordinate is Pixel, the coordinate value starts at 0, and a pixel contains three dot (RGB), with two bytes. The resolution of the device screen is QVGA(240X320).

Length

[in]The horizontal display length.

Width

[in]The horizontal display Width.

Color

[in] The RGB color index value for setting the current foreground color to the specified color. RGB565 palette list to see RGB565 Color Index Table. The RGB565 format is: bit15-bit11 is Red, bit10-bit5 is Green, bit4-bit0 is Blue.

Return Value

Return zero, display the contents in the specified area indicates success. -1 indicates failure.

9. DisplayVertiLine

This function set vertical display on the screen.

Syntax

```
int DisplayVertiLine(
    POINT * prclTrg,
    uint32 Length,
```

```
uint32 Width,
MERCURY_PALETEE_ARRAY_INDEX_E Color
);
```

Parameters

prclTrg

[in] Pointer to a POINT structure that defines the area to be displayed, the unit of coordinate is Pixel, the coordinate value starts at 0, and a pixel contains three dot (RGB), with two bytes. The resolution of the device screen is QVGA(240X320).

Length

[in]the vertical display length.

Width

[in]The vertical display Width.

Color

[in] The RGB color index value for setting the current foreground color to the specified color. RGB565 palette list please to see RGB565 Color Index Table. The RGB565 format is: bit15-bit11 is Red, bit10-bit5 is Green, bit4-bit0 is Blue.

Return Value

Return zero, display the contents in the specified area indicates success. -1 indicates failure.

10. DisplayLineRGB565

This function draws line on the screen by passing in RGB565 value.

Syntax

int DisplayLineRGB565(POINT *p, uint32 Length, uint32 Height, uint16 Color);

Parameters

p

[in] Pointer to a POINT structure that defines the area to be displayed, the unit of coordinate is Pixel, the coordinate value starts at 0, and a pixel contains three dot (RGB), with two bytes. The resolution of the device screen is QVGA(240X320).

Length

[in]The horizontal display length.

Width

[in]The horizontal display Width.

Color

[in]The RGB565 value for line color. The RGB565 format is: bit15-bit11 is Red, bit10-bit5 is Green, bit4-bit0 is Blue.

Return Value

Return zero, display the contents in the specified area indicates success. -1 indicates failure.

11. DisplayPointRGB565

This function draws a point on the screen in real time. When this function is called, the point is displayed directly on the screen without the need of DisplayPaintEnd.

Syntax

int DisplayPointRGB565(POINT *p ,uint16 Color);

Syntax

p

[in] Pointer to a POINT structure that defines the area to be displayed, the unit of coordinate is Pixel, the coordinate value starts at 0, and a pixel contains three dot (RGB), with two bytes. The resolution of the device screen is QVGA(240X320).

Colors

[in] The RGB565 value for point color. The RGB565 format is: bit15-bit11 is Red, bit10-bit5 is Green, bit4-bit0 is Blue.

Return Value

Zero indicates success. -1 indicates failure.

12. DisplayGetMode

This function get the mode supported by the monitor.

Syntax

```
int DisplayGetMode(
    DEVMODEW * prclTrg);
```

Parameters

prclTrg
[out] Pointer to an array of DEVMODEW structures.

Return Value

Equal to zero indicates success.

13. DisplaySetColor

This function sets the current background color and foreground color to the specified color by the color palette index value. The default values of background and foreground colors are all black, so this function must be called after DisplayInit. The RGB565 format: bit15-bit11 is Red, bit10-bit5 is Green, bit4-bit0 is Blue.

Syntax

```
int DisplaySetColor(
    MERCURY_PALETEE_ARRAY_INDEX_E fgColor,
    MERCURY_PALETEE_ARRAY_INDEX_E bgColor
    );
```

Parameters

fgColor

[in]Set the foreground color to display, such as text color. RGB565 palette list please to see RGB565 Color Index Table.

bgColor

[in]Set the background color to display. RGB565 palette list please to see RGB565 Color Index Table.

Return Value

Zero indicates success. -1 indicates failure.

14. LCD_SetColorRGB565

This function sets the current background color and foreground color to the specified color by the specified RGB values. The default values of background and foreground colors are all black, so this function must be called after DisplayInit. The RGB565 format: bit15-bit11 is Red, bit10-bit5 is Green, bit4-bit0 is Blue.

Syntax

```
int LCD_SetColorRGB565(uint16 fgColor, uint16 bgColor);
```

Parameters

fgColor

[in] Set the foreground color to display with the RGB565 value.

bgColor

[in]Set the background color to display with the RGB565 value.

Return Value

Zero indicates success. -1 indicates failure.

15. DisplayClearScreen

The function clears the contents of the screen.

Syntax

```
int DisplayClearScreen(void);
```

Parameters

None.

Return Value

Equal to zero indicates success. -1 indicates failure.

16. DisplaySetBrightness

The function sets the screen backlight brightness. The backlight brightness is recorded with NV, data storage in power dump.

Syntax

```
int DisplaySetBrightness(uint16 Brightness);
```

Parameters

Brightness

[in]Specified the new backlight brightness. The backlight brightness max value is 0xE, if the parameter greater than the max value, set the value of brightness to 0xE.

Brightness is zero, then turn off the screen immediately.

Return Value

Zero indicates success. -1 indicates failure that the screen is off. Otherwise indicates Failed to write NV, see NVITEM_ERROR_E.

17. DisplaySetScreenOffTimeout

The function sets the backlight brightness timeout to the specified timeout. The brightness timeout is recorded with NV, data storage in power dump.

Syntax

int DisplaySetScreenOffTimeout(uint16 Timeout);

Parameters

Timeout

[in]Specified the new screen off timeout, unit is seconds. The range of the screen off timeout value is from 0s to 60s. If timeout is 0, the screen backlight brightness will keep on, and if timeout greater than 60, default setting timeout is 60.

Return Value

Zero indicates success. -1 indicates failure that the screen is off. Otherwise indicates Failed to write NV, see NVITEM_ERROR_E.

18. DisplayGetDirection

This function gets the direction of the screen display.

Syntax

int DisplayGetDirection(DisplayOrientation *Dir);

Parameters

Dir

[out]Pointer to the index of the direction of the screen display.

Return Value

Zero indicates success. -1 indicates failure.

19. DisplaySetDirection

This function sets the direction of the screen display, it cannot be set the direction under the screen off. The display direction is recorded with NV, data storage in power dump.

Syntax

int DisplaySetDirection(DisplayOrientation Dir)

Parameters

Dir

[in]Specifies the direction of the screen display. See DisplayOrientation.

Return Value

Zero indicates success. -1 indicates failure.

20. DisplayScreenOn

This function uses to light screen. Call the function to light up the screen, when the screen is off. When the screen is bright, call this function will reset the screen off timeout.

Syntax

int DisplayScreenOn(void);

Parameters

None.

Return Value

Zero indicates success. -1 indicates failure.

21. DisplayPaintEnd

This function completes the last painting. The LCD display is divided into two steps: First, refresh frame buffer, and then refresh LCD to display buffer.

Previously, Refresh a frame buffer every time, software will automatically refresh LCD to display buffer contents.

Now, After the data of all the frame buffers are refreshed completely, call **DisplayPaintEnd** directly for refresh the LCD to display buffer contents, so as to improve efficiency.

Syntax

int DisplayPaintEnd(void);

Parameters

None.

Return Value

Zero indicates success. -1 indicates failure.

22. DisplayGetRGB565

This function gets RGB565 value by color palette index.

Syntax

uint16 DisplayGetRGB565(MERCURY_PALETEE_ARRAY_INDEX_E paleteIndex);

Parameters

Return RGB565 value.

Devices

UART

1. COM_Init

This function initializes the uart port, configure uart information.

Syntax

```
INT COM_Init(
    ULONG id,
    COM_CONFIG_T* cfg
);
```

Parameters

id

[in]The COM port number which want to initializes, the value of id is 0 or 1.

cfg

[out]Pointer to a structure that the uart port number information. See COM_CONFIG_T.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

2. COM_Deinit

This function de-initializes the uart port.

Syntax

```
INT COM_Deinit(ULONG id);
```

Parameters

id

[in]The port number which already initialized, the value of id is 0 or 1.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

3. COM_Config

This function reconfiguration the specified port number. It will de-initialize the original buffer data before configuration.

Syntax

INT COM_Config(ULONG id, COM_CONFIG_T* cfg);

Parameters

id

[in]the port number, the value of id is 0 or 1.

cfg

[out]Pointer to the structure of including uart port number information. See COM_CONFIG_T

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

4. COM_Read

This function enables an application to receive characters from the UART COM.

Syntax

```
INT COM_Read(
    ULONG id,
    BYTE* pTargetBuffer,
    ULONG BufferLength,
    ULONG* pBytesRead,
    ULONG timeout
);
```

Parameters

id

[in] The com port number, the value of id is 0 or 1.

pTargetBuffer

[in] Pointer to valid memory for read data.

BufferLength

[in] Specifies the size, in bytes, of pTargetBuffer.

pBytesRead

[out] Pointer to a ULONG that contains the number of bytes of read data.

timeout

[in]If read data timeout, stop to read. If timeout equals to zero, this function will return immediately. Unit is ms.

In case of timeout is Nonzero, If the data of the read buffer is greater than **BufferLength**, the data is read directly and returned. Otherwise, the minimum waiting time is 100ms. The waiting time is so calculated: (timeout/100+1) *100ms. So when the range of timeout is 1~99ms, the waiting time is 100ms, and when the range of timeout is 100~199ms, the waiting time is 200ms, and so on.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

5. COM_Write

This function enables an application to transmit bytes to the serial port. And it is a non-block function.

Syntax

```
INT COM_Write(
    ULONG id,
    BYTE* pSourceBytes,
    ULONG NumberOfBytes
    );
```

Parameters

id

[in]The com port number, the value of id is 0 or 1.

pSourceBytes

[in] Pointer to the bytes to be written.

NumberOfBytes

[in] Specifies the number of bytes to be written.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

6. MercuryFastbootStateSet

This function is used to set Fastboot using the USB status.

Syntax

INT MercuryFastbootStateSet(uint32 state);

Parameters

state

[in]The Fastboot uses the USB state. It can set two states, the following table shows the possible values.

Value	Description
FASTBOOT_RUNNING	Fastboot will use the usb.
FASTBOOT_STOP	Fastboot will stop to use the usb.

Return Value

Zero indicates success. -1 indicates failure.

7. MercuryFastbootStateGet

This function gets the Fastboot using the USB status.

Syntax

uint32 MercuryFastbootStateGet(void);

Parameters

None

Return Value

Return the Fastboot using the USB status. The following table shows the possible value.

Value	Description
FASTBOOT_RUNNING = 0	Fastboot is running, USB can't use the AT port.
FASTBOOT_STOP = 1	Fastboot has stopped, USB can transmit data through the AT port.
FASTBOOT_TRANSPARENT0 = 2	Open passthrough from usb to UART0
FASTBOOT_TRANSPARENT1 = 3	Open pssthrough from usb to UART1

8. MercuryUsbRead

This function reads the data through the USB port. The FastBoot must be stopped before calling this function by calling MercuryFastbootStateSet.

Syntax

uint32 MercuryUsbRead(const uint8 *buffer,uint32 length);

Parameters

buffer

[out]Pointer to a buffer that contains the number of bytes of read data.

length

[in]The length of the buffer, in bytes.

Return Value

Nonzero indicates the actual length of the data read. Zero indicates no data can be read or USB port is occupied.

9. MercuryUsbWrite

This function writes the data through the USB port. The FastBoot must be stopped before calling this function by calling MercuryFastbootStateSet.

Syntax

uint32 MercuryUsbWrite(const uint8 *buffer,uint32 length);

Parameters

buffer

[in] Pointer to a buffer that the data will be written in. buffer must be aligned 4.

length

[in] The length of the buffer, in bytes. length must be less than 64 bytes.

Return Value

Nonzero indicates the actual length of the data written. Zero indicates no write data or USB port is occupied.

SPI

1. SPI_Init

This function initializes the spi.

Syntax

```
INT SPI_Init(ULONG id, SPI_CONFIG_S* cfg);
```

Parameters

id
[in] The SPI port number. The range of the id values see MC_SPI_ID_E.

cfg
[out] Pointer to a structure that SPI port information. See SPI_CFG_S.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

2. SPI_Deinit

This function de-initializes the SPI.

Syntax

```
INT SPI_Deinit(ULONG id);
```

Parameters

id
[in] The SPI port number. The range of the id values see MC_SPI_ID_E.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

3. SPI_Config

This function reconfiguration the port number initialized before.

Syntax

```
INT SPI_Config (ULONG id, SPI_CFG* cfg);
```

Parameters

id
[in]the port index of SPI. The range of the id values see MC_SPI_ID_E.

cfg
[out]Pointer to the structure of including SPI port information. See SPI_CFG_S.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

4. SPI_Read

This function enables the device to receive characters by the SPI port.

Syntax

```
INT SPI_Read(
    ULONG id,
    BYTE* pTargetBuffer,
    ULONG BufferLength,
    char* cmd,
    ULONG cmdLen
  );
```

Parameters

id
[in]The port number of SPI.

pTargetBuffer
[out] Pointer of buff of read data.

BufferLength
[in] Specifies the size, in bytes, of read data.

cmd
[in]Send the command before receive data. You must transmit the command to other side first what data you want to read, When you want to read data.

cmdLen
[in]The length, in bytes, of the command.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

5. SPI_Write

This function enables an application to transmit bytes by the SPI port.

Syntax

```
INT SPI_Write(
    ULONG id,
    BYTE* pSourceBytes,
    ULONG NumberOfBytes
    );
```

Parameters

id

[in]The SPI port number.

pSourceBytes

[in] Pointer to the bytes to be written.

NumberOfBytes

[in] Specifies the number of bytes to be written.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

6. SPI_DmaStateSet

This function is used to set whether the SPI is enabled by DMA. This function must be called before **SPI_Init**. Otherwise, the SPI DMA default value is close. And all the spi id must use the same mode.

Syntax

```
INT SPI_DmaStateSet(uint32 enable);
```

Parameters

enable

[in]The enable DMA switch. The following table shows the possible values.

Value	Description
DMA_ENABLE = 1	Enable DMA
DMA_DISABLE = 0	Disable DMA.

Return Value

Zero indicates success. -1 indicates failure.

7. SPI_DmaStateGet

This function is used to get the SPI mode.

Syntax

INT SPI_DmaStateGet(void);

Parameters

None.

Return Value

Return the mode of SPI using.

Battery(charge)

1. BatteryGetStatus

This function obtains the most current battery and power status available on the platform. It fills in the structures pointed to by its parameters.

Syntax

```
INT BatteryGetStatus(
    LPSYSTEM_POWER_STATUS_EX2 pstatus,
    PBOOL pfBatteriesChangedSinceLastCall
);
```

Parameters

pstatus

[out] Pointer to a SYSTEM_POWER_STATUS_EX2 structure. It's only used the **BatteryFlag, BatteryIsExist, BatteryTemperature, BatteryLifePercent, BatteryVoltage** and **ACLineStatus**, All other parameters are reserved.

pfBatteriesChangedSinceLastCall

[out] **Ignored; set to NULL.**

Pointer to a flag that the function sets to TRUE if the user replaced or changed the system's batteries since the last call to this function.

Return Value

Zero indicates success. -1 indicates failure.

NLED

1. NLedInit

This function initializes a notification LED.

Syntax

INT NLedInit(UINT LedID);

Parameters

LedID

[in] The LED number, please refer to the NLED_ID_E for the range of LedID values.

Return Value

Equal to zero indicates success. -1 indicates failure.

2. NLedDeinit

This function de-initializes a notification LED, release led resources.

Syntax

INT NLedDeinit(UINT LedID)

Parameters

LedID

[in] The LED number, please refer to the NLED_ID_E for the range of LedID values.

Return Value

Equal to zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

3. NLedSetMode

This function set the working mode of the notification Led.

Syntax

```
INT NLedSetMode(
    UINT LedID,
    INT OffOnBlink,
```

```
        LONG TotalCycleTime,
        LONG OnTime,
        LONG OffTime,
        INT MetaCycleOn,
        INT MetaCycleOff
    );
```

Parameters

LedNum

[in] LED number, the range of LedID values refer to the NLED_ID_E.

OffOnBlink

[in] Current setting. If it is not equal to NLED_MODE_BLINK, The other parameters below can be set to 0.

The following table shows the defined values.

Value	Description
NLED_MODE_OFF	Off
NLED_MODE_ON	On
NLED_MODE_BLINK	Blink

TotalCycleTime

[in]Total cycle time of a blink, in milliseconds. In Blink mode, the parameter cannot be set to 0.

OnTime

[in]On time of the cycle, in milliseconds. In Blink mode, the parameter cannot be set to 0.

OffTime

[in]Off time of the cycle, in milliseconds. In Blink mode, if **Offtime** equals 0, then the LED will be bright always until the end of the total time.

MetaCycleOn

[in]Number of on blink cycles. In Blink mode, the parameter cannot be set to 0.

MetaCycleOff

[in]Number of off blink cycles. In Blink mode, if **MetaCycleOff** equals 0, then the LED will be bright always until the end of the total time.

Return Value

Equal to zero indicates success, -1 indicates failure.

TTS

1. TTS_Init

The function initializes the TTS module. This function allocates the heap, output buffer, and set the initial value of speed, read digit number, volume, pitch, channel, etc.

Syntax

int TTS_Init();

Parameters

None.

Return Value

Equal to zero indicates success. -1 indicates failure. To get extend error information, call GetLastError.

2. TTS_Deinit

The function de-initializes the TTS module. This function releases the allocated resources.

Syntax

int TTS_Deinit();

Parameters

None.

Return Value

Equal to zero indicates success. -1 indicates failure.

3. TTS_SetParams

This function sets the speak speed and pitch of the TTS.

Syntax

int TTS_SetParams(TTS_PARAM_S* mode);

Parameters

mode

[in]Pointer to a TTS_PARAM_S structure that contains speak speed and pitch of TTS.

Return Value

Equal to zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

4. TTS_GetParams

The function gets the speak speed and pitch of TTS.

Syntax

int TTS_GetParams(TTS_PARAM_S* mode);

Parameters

mode

[out] Pointer to a TTS_PARAM_S structure that TTS current speak speed and pitch.

Return Value

Equal to zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

5. TTS_PlayText

This function play the text by speech. If one text is already playing, the other one will not be able to play until the first one is aborted.

Syntax

int TTS_PlayText(const uint8 * text, int len);

Parameters

text

[in]Pointer to the text that the string will be played.

len

[in] the size, in bytes, of the text to play.

Return Value

Equal to zero indicates success. -1 indicates failure.

To get extend error information, call GetLastError. The following table shows the possible values.

Value	Description
ERROR_SUCCESS 0	Success
ERROR_TTS_INVALID_HANDLE 0x80100001	tts handle create fail or no create
ERROR_TTS_PLAYING 0x80100003	This means that a audio is already playing, and now, the other one can't to play until the first one is aborted.
ERROR_TTS_OVER_TEXT_LEN 0x80100004	over max text length, the max text length is TTS_MAX_TEXT_LEN(958 bytes)

6. TTS_Abort

This function stops the TTS to play.

Syntax

int TTS_Abort();

Parameters

None.

Return Value

Equal to zero indicates success. -1 indicates failure.

7. PCM_StartPlay

This function starts to play ext pcm data(16khz 16bit).

Syntax

int PCM_StartPlay();

Parameters

None.

Return Value

Equal to zero indicates success. -1 indicates failure.

8. PCM_FillData

This function fills the pcm data.

Syntax

int PCM_FillData(uint8 * chBuf, uint32 nSize);

Parameters

chBuf
[in]Pointer to a buffer that the pcm data will be filled.
nSize
[in]The size, in byte, of the pcm data that will be filled. When the data is completed, you need to call PCM_FillData(chBuf, 0) to tell the system that the data has been completed.

Return Value

Equal to zero indicates success. -1 indicates failure. To get extend error information, call GetLastError.

9. PCM_StopPlay

Thist function stops to play pcm data.

Syntax

int PCM_StopPlay();

Parameters

None.

Return Value

Return zero always.

10. AMR_StartPlay

This function starts to play amr format data.

Syntax

int AMR_StartPlay(uint8* pucData, uint32 uiDataLength);

Parameters

pucData
[in] Pointer to a buffer that the amr data will be played.

uiDataLength
[in] The size, in byte, of the amr data.

Return Value

Equal to zero indicates success. -1 indicates failure.

11. AMR_StopPlay

Thist function stops to play amr format data.

Syntax

int AMR_StopPlay();

Parameters

None.

Return Value

Return zero always.

2D Bar

1. CAM_Init

This function initializes camera sensor hardware. When this function is called, the camera driver detects and initializes the hardware, allocates and initializes its data structures, and entry camera preview interface.

Syntax

DCAMERA_RETURN_VALUE_E Cam_Init(void);

Parameters

NONE

Return Value

DCAMERA_OP_SUCCESS indicates success, otherwise indicates failure. See DCAMERA_RETURN_VALUE_E.

2. Cam_DeInit

This function de-initializes the camera sensor hardware.

Syntax

DCAMERA_RETURN_VALUE_E Cam_DeInit (void);

Parameters

NONE

Return Value

DCAMERA_OP_SUCCESS indicates success, otherwise indicates failure. See DCAMERA_RETURN_VALUE_E.

3. Cam_StartScan

This function is used for camera start to scan the bar code. The scan result will be reported via message notification mode.

The message notification mode is that app register NotifyCallback function, the platform will notify the app of various messages through NotifyCallback .

Note: The data successfully reported by scan is a SYMBOL_RESULT_T structure. The valid character length is datalen, so you can print and display these valid characters only.

Syntax

```
DCAMERA_RETURN_VALUE_E Cam_StartScan(
    BARSCAN_MODE_VALUE_E mode
);
```

Parameters

mode

[in]The bar scan mode. If equal to 0, it means continuous scan. 1 means one-time scans. 2 means high resolution motion detection scanning mode. 3 means only scan one-dimensional codes. 4 means low resolution motion detection scanning mode. Otherwise it will return to parameter error. See BARSCAN_MODE_VALUE_E.

Return Value

DCAMERA_OP_SUCCESS indicates success, otherwise indicates failure. See DCAMERA_RETURN_VALUE_E.

when return error, you need call **Cam_DeInit** first, then call **Cam_Init**, and then call **Cam_StartScan** to rescan again.

4. Cam_AbortScan

This function pauses the camera scan. This function is called after Cam_StartScan.

Syntax

```
DCAMERA_RETURN_VALUE_E Cam_AbortScan(void)
```

Parameters

None;

Return Value

DCAMERA_OP_SUCCESS indicates success, otherwise indicates failure. See DCAMERA_RETURN_VALUE_E.

5. Cam_Suspend

This function suspends the scan code, and can directly call **Cam_StartScan** to continue the scan code, without initializing the hardware.

Syntax

```
DCAMERA_RETURN_VALUE_E Cam_Suspend(void);
```

Parameters

None;

Return Value

DCAMERA_OP_SUCCESS indicates success, otherwise indicates failure. See DCAMERA_RETURN_VALUE_E.

6. Cam_QR_Enc

This function encodes string to QRcode in Bit matrix.

Syntax

QR_ENC_CODE_T* Cam_QR_Enc(int lenth, const unsigned char* data);

Parameters

lenth

[in]The actual size, in bytes, of the data that will to be encoded. The encode length may be smaller than the size of the **data** parameter.

data

[in]Pointer to the data to be encoded.

Return Value

Pointer to a QR_ENC_CODE_T structure containing encode result information. After this data is not used again, we must manually release two pointer resources: the bits pointer in QR_ENC_CODE_T first, and then the QR_ENC_CODE_T itself.

7. Cam_FeatureConfig

This function is used to configure the feature mode of camera support.

Syntax

```
DCAMERA_RETURN_VALUE_E Cam_FeatureConfig(
    BARSCAN_FEATURE_CONFIGURE_E feature2Config
    );
```

Parameters

feature2Config

[in]Configuration camera supported functional mode. See

BARSCAN_FEATURE_CONFIGURE_E.

Return Value

DCAMERA_OP_SUCCESS indicates success, otherwise indicates failure. See DCAMERA_RETURN_VALUE_E.

8. Cam_QR_SetEncLEVEL

This function can set the fault tolerance level for generating QR code.

Syntax

int Cam_QR_SetEncLEVEL(QRecLevel level);

Parameters

level

[in]The level of error correction for generating QR codes. See QRecLevel.

Return Value

Return zero always.

9. Cam_QR_SetScanDensity

This function sets the scan density of QR code. The default value is 1, the density can be set according to the application scenario. The larger the density, the faster the decoding speed of angle-free or small-angle QR codes. One-dimensional code and QR code are in the same frame, Density 2 is recommended.

Syntax

int Cam_QR_SetScanDensity(int xDensity, int yDensity);

Parameters

xDensity

[in]The density of x-axis, the default value is 1.

yDensity

[in] The density of y-axis, the default value is 1.

Return Value

Return zero always.

10. Cam_QR_GetScanDensity

This function gets the QR code density.

Syntax

int Cam_QR_GetScanDensity(int *pXDensity,int *pYDensity);

Parameters

xDensity
[out]The density of x-axis, the default value is 1.
yDensity
[out] The density of y-axis, the default value is 1.

Return Value

Return zero always.

11. Cam_CfgSymbolEnable

This function configs symbol decode enable state.

Syntax

int Cam_CfgSymbolEnable(AMOI_SYMBOL_TYPE_T symbol,int isEnable);

Parameters

symbol
[in] The symbol of bar code. See AMOI_SYMBOL_TYPE_T. The default support symbols are Code 39, QR code, Code 128, GOODS(EAN ISBN UPCA etc).
isEnable
[in]1 indicates enable, 0 indicates disable.

Return Value

0 indicates success, -1 indicates failure.

12. Cam_GetBarLibVersion

This function gets the scan library version.

Syntax

char * Cam_GetBarLibVersion();

Parameters

None

Return Value

Return the scan library version.

13. Cam_SetPrescanLine

This function sets the prescan line specification.

Syntax

int Cam_SetPrescanLine(int type);

Parameters

type
[in]The prescan line type. Its value may be 0(5*7) or 1(9*13). The default type is 0.

Return Value

0 indicates success, -1 indicates failure.

GPIO(interrupt)

1. GIO_Init

This function initialize configuration of the GPIO. Before call another GPIO functions, it must be call the GIO_Init function first.

Syntax

DWORD GIO_Init(void)

Parameters

None.

Return Value

Zero indicates success, -1 indicates failure.

2. GIO_Deinit

This function de-initialize GPIO function.

Syntax

BOOL GIO_Deinit(void)

Parameters

None.

Return Value

Zero indicates success, -1 indicates failure.

3. GIO_SetBit

This function set the GPIO bit to 1.

Syntax

VOID GIO_SetBit(DWORD id)

Parameters

id

[in]the GPIO pin id. The range of the id values are 37, 38, 39, 56, 57, 68, 69, 4, 5.

Return Value

None.

4. GIO_ClrBit

This function clear the GPIO bit to 0.

Syntax

```
VOID GIO_ClrBit(DWORD id);
```

Parameters

id

[in]the GPIO pin id. The range of the id values are 37, 38, 39, 56, 57, 68, 69, 4, 5.

Return Value

None.

5. GIO_GetBit

This function gets the value of an GPIO. This function can be called in the INPUT mode only.

Syntax

```
DWORD GIO_GetBit(DWORD id)
```

Parameters

id

[in]the GPIO pin id. The range of the id values are 37, 38, 39, 56, 57, 68, 69, 4, 5.

Return Value

The value of the GPIO state indicates success, others indicates failure.

6. GIO_SetMode

This function set the mode for the GPIO pin number.

Syntax

```
VOID GIO_SetMode(
    DWORD id,
    MERCURY_GPIO_CFG_S* pCfg);
```

Parameters

id

[in]the GPIO pin id. The range of the id values are 37, 38, 39, 56, 57, 68, 69, 4, 5.

pCfg

[in]The mode configuration for the GPIO pin. See MERCURY_GPIO_CFG_S.

Return Value

None.

NV

1. NV_Init

This function initialize the NV.

Syntax

int NV_Init(void);

Parameters

None.

Return Value

Equal to zero indicates success, -1 indicates failure.

2. NV_Deinit

This function de-initialize the NV.

Syntax

int NV_Deinit(void);

Parameters

None.

Return Value

Equal to zero indicates success, -1 indicates failure.

3. NV_Read

This function read an item from the medium.

Syntax

int NV_Read(WORD ItemID, WORD cchSize, BYTE *pBuf);

Parameters

ItemID

[in]The identifier of the NV item to be read, the value is from 0 to 499. If the item is not created before, the function will return to error.

cchSize

[in]Size in bytes to be read.

pBuf
[out] Pointer to the buffer to hold the data read.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.
The following table shows possible error values.

Value	Description
ERROR_NV_NOT_EXIST	0x800a0004. The NV item does not exist.
ERROR_INVALID_PARAMETER	0x800a0002. Parameters are invalid, e.g. buf_ptr is NULL, count is larger than the size of this NV item or the specified Identifier does not exist.

4. NV_Write

This function write an item to the medium. If the item does not exist, it will be created.

Syntax

```
int NV_Write(WORD ItemID, WORD cchSize, BYTE *pBuf);
```

Parameters

ItemID
[in]Identifier of the NV item to be written, the value is from 0 to 499.

cchSize
[in]Size in byte of this item. If the item is exist and the cchSize parameter is larger than the original size of the item, error will be returned.

pBuf
[in]Pointer to the buffer holding the data of the item.

Return Value

Equal to zero indicates success, other indicates failure. To get extended error information, call GetLastError.
The following table shows possible values.

Value	Description
ERROR_NV_NO_ENOUGH_RESOURCE	0x800a0003. There is no enough resource to complete this operation, e.g. no enough space on the medium.

ERROR_INVALID_PARAMETER	0x800a0002. Parameters are invalid, e.g. buf_ptr is NULL or Identifier is invalid.

5. NV_Delete

This function delete an item in the range.

Syntax

int NV_Delete(WORD ItemID);

Parameters

ItemID
[in]Identifier of the item to be deleted, the value is in the range of 0 to 499.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

6. NV_AppInfoRead

This function read the app information from the medium.

Syntax

int NV_AppInfoRead(void* pAppInfo, DWORD len);

Parameters

pAppInfo
[out] Pointer to a buffer that saves the app information.

len
[in]Size, in bytes, of the buffer pointed to by pAppinfo. And the size of the pAppInfo must equal to 640 bytes.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call the GetLastError function.

7. NV_AppInfoWrite

This function write the app information to the medium.

Syntax

int NV_AppInfoWrite(void* pAppInfo, DWORD len);

Parameters

pAppInfo

[in] Pointer to a buffer that saves the app information.

len

[in]Size, in bytes, of the buffer pointed to by pAppinfo. And the size of the pAppInfo must equal to 640 bytes.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call the GetLastError function.

User Inputs

Keypad

1. KP_Init

This function initialize keypad.

Syntax

INT KP_Init(void)

Parameters

None.

Return Value

Zero indicates success, -1 indicates failure.

2. KP_Deinit

This function de-initialize all keypad.

Syntax

INT KP_Deinit(void);

Parameters

None.

Return Value

Zero indicates success, -1 indicates failure.

3. KP_RegisterApp

This function register the specified app message.

Syntax

INT KP_RegisterApp(UINT uid, LPVOID kpCallback);

Parameters

uid

[in]The keypad id, the value must be in the enumerations KEYPAD_UID_E.

kpCallback
[in]Pointer to a callback of the KeypadCallBackFunc to be called until successful register.

Return Value

Zero indicates success, -1 indicates failure.

4. KP_DeregisterApp

This function free the specified app message.

Syntax

INT KP_DeregisterApp(UINT uid)

Parameters

uid
[in]The keypad id, the value must be in the enumerations KEYPAD_UID_E.

Return Value

Equal to zero indicates success, -1 indicates failure.

5. KP_SetFocus

This function sets the keyboard focus to the specified window. All subsequent keyboard input is directed to this window. The window, if any, that previously had the keyboard focus loses it. This function must be called after **KP_RegisterApp**.

Syntax

INT KP_SetFocus(UINT uid)

Parameters

uid
[in]The keypad id, the value must be in the enumerations KEYPAD_UID_E.

Return Value

Equal to zero indicates success, -1 indicates failure.

6. KP_SetKeyPressSound

This function set the keypad whether play input sound. This function is not

implemented at the moment.

Syntax

INT KP_SetKeyPressSound(BOOL bEnabled);

Parameters

bEnabled

[in]input sound status. true indicates play sound, false indicates not play sound.

Return Value

Equal to zero indicates success, -1 indicates failure.

7. KeypadCallBackFunc

This function is a callback, it will be called when successful register app.

Syntax

VOID KeypadCallBackFunc(UINT uid,UINT singleCode,UINT keyCode)

Parameters

uid

[in]the keypad uid, the value must be in the enumerations KEYPAD_UID_E.

singleCode

[in]The key single code, such as key press down and up.

keyCode

[in] The specified virtual key code value. Reference Virtual Key Code definition.

Return Value

None.

8. KP_SetBlackLight

This function can set the keypad blacklight brightness. The blacklight closed by default.

Syntax

INT KP_SetBlackLight(UINT brightness);

Parameters

brightness

[in]The brightness of keypad. The value of backlight brightness is 0 to 15. If brightness equals to 0, blacklight is close. And if it greaters than 15, it will return to fail.

Return Value

Equal to zero indicates success, -1 indicates failure.

Power Management

1. WakeLock

This function is used to add a new lock. You can add several locks at the same time. This function and **WakeUnlock** are used for sleep mode, For details, please refer to the document: << AC35 休眠应用文档.pdf >>.

Syntax

```
int    WakeLock(
    WAKE_LOCK_MODES LockMode,
    UINT8 * LockName
);
```

Parameters

LockMode
[in] The mode of the new lock. See WAKE_LOCK_MODES

LockName
[in] The name of the new lock.

Return Value

Greater than or equal to zero indicates success, less than zero failure.

2. WakeUnlock

This function is used to release a effective lock.

Syntax

```
int WakeUnlock(WAKE_LOCK_MODES LockMode, UINT8 * LockName);
```

Parameters

LockMode
[in] The mode of the lock. See WAKE_LOCK_MODES

LockName
[in] The lock that want to release which had created in WakeLock.

Return Value

Equal to zero indicates success, less than zero failure.

3. PowerOff

This function is used to power off device.

Syntax

```
int    PowerOff(void);
```

Parameters

None.

Return Value

Zero indicates success. -1 indicates failure.

4. PowerReboot

This function is used to reboot device.

Syntax

```
int    PowerReboot(void);
```

Parameters

None.

Return Value

Zero indicates success. -1 indicates failure.

5. Power_GetVoltagetoPercent

This function gets the battery current level.

Syntax

```
uint32 Power_GetVoltagetoPercent(void);
```

Parameters

None.

Return Value

Return to the battery level, the range of values for the level is 0 to 100.

6. ADC_GetResult

This function gets the scale value by ADC. The resolution accuracy of ADC is 12 bits, so the range of the value of ADC is 0~2^{12}-1, then the corresponding voltage is 0~3V(depending on the reference voltage).

Syntax

```
int ADC_GetResult(ADC_ID_E adcID, ADC_SCALE_E adcScale);
```

Parameters

adcID
[in]The ADC id. See ADC_ID_E.

adcScale
[in]The supported ADC scales. See ADC_SCALE_E.

Return Value

Return scale value, the value is in the range 0 through 4095.

7. SetChgOverHighTemp

This function sets charging high temperature threshold. Please select the appropriate temperature according to our **Temperature coefficient table** and the customer's own battery characteristics.

Syntax

```
void SetChgOverHighTemp(uint16 highTemp);
```

Parameters

highTemp
[in]The charging high temperature threshold.

Return Value

None.

8. SetChgOverLowTemp

This function sets charging low temperature threshold. Please select the appropriate temperature according to our **Temperature coefficient table** and the customer's own battery characteristics.

Syntax

void SetChgOverLowTemp(uint16 lowTemp);

Parameters

lowTemp
[in] The charging low temperature threshold.

Return Value

None.

9. SetRechgVol

This function set the threshold value of recharge voltage.

Syntax

void SetRechgVol(uint16 rechgVol);

Parameters

rechgVol
[in] The threshold value of recharge voltage, unit is mV.

Return Value

None.

10. SetChgEndVol

This function set the end of charge voltage value.

Syntax

void SetChgEndVol(uint16 chg_end_vol);

Parameters

chg_end_vol
[in]The terminal charging voltage.

Return Value

None.

11. SetChgSwtich

This function sets the charging switch.

Syntax

int SetChgSwtich(SWITCH_STATE_E chgSwitch);

Parameters

chgSwitch

[in]The charging switch. The following table shows the possible value.

Value	Description
DEFAULT_AUTO = 0	Use the default value
MANUAL_SWITCH_ON=1	Open the charging switch
MANUAL_SWITCH_OFF=2	Close the charging switch

Return Value

Zero indicates the function call is successful, -1 indicates failure.

12. SetAutoIdentAdp

This function sets whether the adapter is automatically identified.

Syntax

int SetAutoIdentAdp(BOOLEAN autoIdentity);

Parameters

autoIdentity

[in]Configure whether the adapter is automatically identified.

Return Value

Zero indicates successful, -1 indicates failure.

13. GetAdpType

The function gets the adapter type.

Syntax

int GetAdpType();

Parameters

None.

Return Value

Return the adapter type. The following table shows the possible values.

Value	Description
CHGMNG_ADP_UNKNOW = 0	Unknow the adapter type.
CHGMNG_ADP_STANDARD=1	The standard adapter.
CHGMNG_ADP_NONSTANDARD=2	Nonstandard adapter.
CHGMNG_ADP_USB = 3	USB charge

PWM

1. MercuryPWM_Init

This function initializes the PWM. You must initialize before you call other PWM related functions.

Syntax

void MercuryPWM_Init(void);

Parameters

None.

Return Value

None.

2. MercuryPWM_Config

This function is used to configure frequency and duty cycle.

Syntax

void MercuryPWM_Config(uint32 freq, uint16 duty_cycle);

Parameters

freq

[in]Frequency, in HZ, of PWM. The PWM frequency range for module support is 1~2978HZ.

duty_cycle

[in]Duty cycle of PWM. The range of the duty_cycle is 0~100.

Return Value

None.

3. MercuryPWM_Start

This function starts playing the buzzer. After the function is executed, the level will appear at a high or low level at random. The processing method is that you can call the **MercuryPWM_Config** to set the duty cycle to 0(or 100), then call **MercuryPWM_Start** again for a shorter time, so that we can control the location at low level(or at high level).

Syntax

void MercuryPWM_Start(uint32 times);

Parameters

times

[in]The PWM hold time. Its unit is 10ms. if times is zero, PWM will run always until MercuryPWM_Stop been called.

Return Value

None.

4. MercuryPWM_Stop

This function stops PWM to run.

Syntax

void MercuryPWM_Stop(void);

Parameters

None.

Return Value

None.

5. MercuryPWM_Deinit

This function de-initializes PWM.

Syntax

void MercuryPWM_Deinit(void);

Parameters

None.

Return Value

None.

File System

Note:

Currently, the file system supports the E disk only. If the application had been doing writes to the file, before calling **CloseHandle**, you have to call **FlushFileBuffers** function. And if you do not call **CloseHandle** after write a file, you must call the **FlushFileBuffers** function before calling **UnInitFileSystem** or power off.

1. InitFileSystem

This function initializes file system before use file system api. All the other api interfaces should be called after this function. This function cannot be used in conjunction wiht **InitFileSystemPlus** in the same Application.

The maximum capacity of the file system is theoretically supported by 16M, but only to ensure the stability of 8M, the excess part of the stability cannot be guaranteed.

Syntax

INT InitFileSystem (uint32 startAddr, FILESYS_CAPACITY_E cap);

Parameters

startAddr

[in]This parameter represents the index of the file system sector, and the one sector is 4K. Such as File system starts from at 2M, startAddr equals 512 (2M/4K = 512).

CAPACITY

[in] The filesystem capacity, please select the right capacity according to the size of the flash. See FILESYS_CAPACITY_E.

Such as if the total capacity of SPI flash is 4M, **CAPACITY** can't be greater than 4, and the value of the **startAddr** parameter ranges from 0 to 1023.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

2. InitFileSystemPlus

This function initializes filesystem with specific parameter settings before use filesystem api. All the other api interfaces should be called after this function. This function cannot be used in conjunction wiht **InitFileSystem** in the same Application.

The maximum capacity of the file system is theoretically supported by 16M, but only to ensure the stability of 8M, the excess part of the stability cannot be guaranteed.

Syntax

```
int InitFileSystemPlus(
        uint32 startAddr,
        FILESYS_CAPACITY_E fileSysCap,
        FILESYS_CAPACITY_E spiFlashCap
    );
```

Parameters

startAddr

[in]This parameter represents the index of the file system sector, and the one sector is 4K. Such as File system starts from at 2M, startAddr equals 512 (2M/4K = 512).

fileSysCap

[in] The filesystem capacity, please select the right capacity according to the size of the flash, this parameter must be less than or equals to **spiFlashCap.** See FILESYS_CAPACITY_E.

spiFlashCap

[in]The total capacity of SPI flash. See FILESYS_CAPACITY_E.

Such as if the **spiFlashCap** equals 8M, **fileSysCap** must be less than or equals to 8, and if **fileSysCap** equals 8M, the value of the **startAddr** parameter ranges from 0 to 2047.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

3. UnInitFileSystem

This function de-initializes file system.

Syntax

INT UnInitFileSystem(void);

Parameters

None.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

4. CreateDirectory

This function creates a new directory. If the underlying file system supports security on files and directories, the function applies a specified security descriptor to the new directory.

Currently, the file system supports the E disk only.

Syntax

```
INT CreateDirectory(
    LPCTSTR lpPathName,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
```

Parameters

lpPathName

[in] The name of path you want to create ,it must be Unicode string.

lpSecurityAttributes

[in] Ignored; set to NULL.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

5. DeleteDirectory

This function deletes a directory. When there is a file in a folder, you can't delete the folder directly. You need to empty the file in the folder before you can delete the folder.

Syntax

INT DeleteDirectory(LPCTSTR lpPathName);

Parameters

lpPathName

[in] The name of path you want to delete ,it must be Unicode string.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

6. CreateFile

This function creates or opens a file. It returns a handle to access the object.

Currently, the file system supports the E disk only.

Syntax

```
HANDLE CreateFile(
    LPCTSTR lpFileName,
    DWORD dwDesiredAccess,
    DWORD dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD dwCreationDisposition,
    DWORD dwFlagsAndAttributes,
    HANDLE hTemplateFile
);
```

Parameters

lpFileName

[in] Pointer to the name of path of file ,it must be Unicode string .

dwDesiredAccess

[in] Type of access to the file. When a file opened and created, one must be used and only one access mode is constant, at least one operation constant.

The following table shows possible values.

Value	Description
MCFILE_ACCESS_MODE_CREATE_NEW	Creates a new file. The function fails if the specified file already exists.
MCFILE_ACCESS_MODE_CREATE_ALWAYS	Creates a new file. If the file exists, delete this file first, then create a new file.
MCFILE_ACCESS_MODE_OPEN_EXISTING	Opens the file. The function fails if the file does not exist.
MCFILE_ACCESS_MODE_OPEN_ALWAYS	Opens the file, if it exists. If the file does not exist, the function creates the file as if this parameter were set to CREATE_NEW.
MCFILE_ACCESS_MODE_APPEND	Opens the file, if it exists. If the file does not exist, the function creates the file. On the way to open the file, All the data are only increase at the end of the file when writing. Repositioning is no use, only to write from the end.
MCFILE_OPERATE_MODE_READ	Specifies read access to the object. Data can be read from the file, and the file pointer can be moved. Combine with MCFILE_OPERATE_MODE_WRITE for read/write access.
MCFILE_OPERATE_MODE_WRITE	Specifies write access to the object. Data can be written to the file, and the file pointer can be moved. Combine with MCFILE_OPERATE_MODE_READ for read/write access.
MCFILE_OPERATE_MODE_SHARE_READ	(Reserve)no use
MCFILE_OPERATE_MODE_SHARE_WRITE	(Reserve)no use

dwShareMode

[in]set to 0;

lpSecurityAttributes

[in] set to NULL.

dwCreationDisposition

[in] set to 0

dwFlagsAndAttributes

[in]set to 0.

hTemplateFile

[in]set to 0.

Return Value

None zero indicates success, it is an open handle to the specified file. Equal to zero indicates failure.

7. DeleteFile

This function deletes a file from file system.

Syntax

```
INT DeleteFile(
    LPCTSTR lpFileName
);
```

Parameters

lpFileName
[in] Pointer to the name of path of file , it must be Unicode string.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

8. FindFirstFile

This function searches a directory for a file or subdirectory with the specified file name.

Syntax

```
HANDLE FindFirstFile(
    LPCTSTR lpFileName,
```

```
    LPMCFILE_FIND_DATA_T    lpFindFileData
);
```

Parameters

lpFileName

[in] Pointer to the file name you want to find, such as L"E:*.mp3", L"E:*.3gp", L"E:\\test*", or L"E:\\123.mp3"...

lpFindFileData

[out] If file is founded ,the information of file is stored in this structure. See LPMCFILE_FIND_DATA_T.

Return Value

Nonzero indicates success, the file founded ,this find handle can be used to find next file match the parameter ' lpFileName '. Equal to zero indicates failure.

9. FindNextFile

This function continues a file search from a previous call to the FindFirstFile.

Syntax

```
INT FindNextFile(
    HANDLE hFindFile,
    LPMCFILE_FIND_DATA_T lpFindFileData
);
```

Parameters

hFindFile

[in] The search handle that is returned from a previous call to the FindFirstFile function.

lpFindFileData

[out] If file is founded ,the information of file is stored in this structure. See LPMCFILE_FIND_DATA_T.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

10. FindClose

This function closes the specified search handle. The FindFirstFile and the FindNextFile functions use the search handle to locate files with names with a

specified name.

Syntax

```
INT FindClose(
    HANDLE hFindFile
);
```

Parameters

hFindFile
[in] Search handle. This handle must have been previously opened by the FindFirstFile function.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

11. FlushFileBuffers

This function clears the buffers for the specified file and writes all buffered data to the file. The **WriteFile** function typically writes data to an internal buffer, this function writes the buffered information for the specified file to disk.

Syntax

```
INT FlushFileBuffers(
    HANDLE hFile
);
```

Parameters

hFile
[in] Handle to an open file. The function flushes this file's buffers. The file handle must have **MCFILE_OPERATE_MODE_WRITE** access to the file.
If this parameter is a handle to a communications device, the function only flushes the transmit buffer.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

12. GetFileSize

This function obtains the size, in bytes, of the specified file.

Syntax

```
DWORD GetFileSize(
    HANDLE hFile,
    LPDWORD lpFileSizeHigh
);
```

Parameters

hFile
[in] Open handle to the file whose size is being returned. The handle must have been created with either **MCFILE_OPERATE_MODE_READ** or **MCFILE_OPERATE_MODE_WRITE** access to the file.

lpFileSizeHigh
[out] Ignored. Set to NULL.

Return Value

The file size indicates success. -1 indicates failure. To get extended error information, call GetLastError.

13. SetFileSize

This function sets file length. If the size of the set is greater than the original file length, then fill 0 at the end of the file; otherwise, delete the deleted part of the file.

Syntax

```
int SetFileSize(HANDLE hFile, uint32 size);
```

Parameters

hFile
[in]The handle returned by CreateFile.

size
[in] The length ,in bytes, you want to set. If parameter **size** is less than file length ,some data in the tail of file will be lost.

Return Value

Zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

14. ReadFile

This function reads data from a file, starting at the position indicated by the file pointer. After the read operation has been completed, the file pointer is adjusted by the number of bytes read.

Syntax

```
INT ReadFile(
    HANDLE hFile,
    LPVOID lpBuffer,
    DWORD nNumberOfBytesToRead,
    LPDWORD lpNumberOfBytesRead,
    LPOVERLAPPED lpOverlapped
);
```

Parameters

hFile
[in] The handle returned by CreateFile.

lpBuffer
[out] Pointer to the buffer that the data has be read will stored in this buffer

nNumberOfBytesToRead
[in] The number you want to read ,unit is byte.

lpNumberOfBytesRead
[out] Pointer to the number of bytes read.

lpOverlapped
[in] Unsupported. Set to NULL.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

15. RenameFile

This function renames a file.

Syntax

```
int RenameFile(const uint16 *sour,const uint16 *dest);
```

Parameters

sour
[in] The name of path and source file ,it must be Unicode string.
dest
[out] The name of path and destination file ,it must be Unicode string.

Return Value

Zero indicates success. Others indicates failure. To get extended error information, call GetLastError.

16. SetFilePointer

This function sets current position in file.

Syntax

```
INT SetFilePointer(
    HANDLE hFile,
    LONG lDistanceToMove,
    PLONG lpDistanceToMoveHigh,
    DWORD dwMoveMethod
);
```

Parameters

hFile
[in] The handle returned by CreateFile.

lDistanceToMove
[in] The relative position of origin. A positive value for this parameter moves the file pointer forward in the file, and a negative value moves the file pointer backward. You cannot use a negative value to move back past beyond the beginning of a file.

lpDistanceToMoveHigh
[in]ignored, set to NULL.

dwMoveMethod
[in] the absolutely you will be seek. The following table shows possible values.

Value	Description
MCFILE_SEEK_BEGIN	Indicates that the starting point is zero or the beginning of the file.
MCFILE_SEEK_CUR	Indicates that the starting point is the current value of the file pointer.
MCFILE_SEEK_END	Indicates that the starting point is the current

	end-of-file position.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

17. GetFilePointer

This function gets current position in file.

Syntax

int GetFilePointer(HANDLE hFile, uint32 origin, INT* currentpos);

Parameters

hFile

[in] The handle returned by CreateFile.

origin

[in] The absolutely you will be find. It can be:

Value	Description
MCFILE_SEEK_BEGIN	0, Returns the offset relative to the start position of the file.
MCFILE_SEEK_END	2, Returns the offset relative to the end position of the file.

currentpos

[out] If api success ,this will stored the relative position of origin.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

18. WriteFile

This function writes data to a file, starting at the position indicated by the file pointer. After the write operation has been completed, the file pointer is adjusted by the number of bytes written.

Syntax

```
INT WriteFile(
    HANDLE hFile,
```

```
    LPCVOID lpBuffer,
    DWORD nNumberOfBytesToWrite,
    LPDWORD lpNumberOfBytesWritten,
    LPOVERLAPPED lpOverlapped
);
```

Parameters

hFile
[in] The handle returned by CreateFile.

lpBuffer
[in] Pointer to the buffer that the data will be written is stored in this buffer.

nNumberOfBytesToWrite
[in] Number of bytes to write to the file.
A value of zero specifies a null write operation. A null write operation does not write any bytes, but does cause the time stamp to change. This function does not truncate the file.

lpNumberOfBytesWritten
[out] Pointer to the number of bytes written by this function call. This function sets this value to zero before taking action or checking errors.

lpOverlapped
[in] Unsupported. Set to NULL.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

19. CloseHandle

This function closes an open object handle.

Syntax

```
INT CloseHandle(
    HANDLE hFile
);
```

Parameters

hFile
[in] Handle to an open file.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

20. FomatDisk

This function formats the disk. It needs about 10s.

Syntax

```
INT FomatDisk(void);
```

Parameters

None.

Return Value

Zero indicates success. -1 indicates failure.

21. GetDeviceFreeSpace

This function gets the disk free space.

Syntax

```
INT GetDeviceFreeSpace(uint32 * pDiskFreeSize);
```

Parameters

pDiskFreeSize

[out] Pointer to a variable to receive the number of free bytes on the disk.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

22. GetDeviceUsedSpace

This function gets the disk space that has been used.

Syntax

```
INT GetDeviceUsedSpace(
        uint32 * pDiskUsedSize
        );
```

Parameters

pDiskUsedSize
[out] Pointer to a variable to receive the number of used bytes on the disk.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

Little File System

NOTE: LittleFS is not support upgrade by FOTA, you have to upgrade by FOTA with the original file system.

1. LittlefsInit

This function initializes littlefs with specific parameter settings before use littlefs api. All the other api interfaces should be called after this function.

The maximum capacity of the file system is theoretically supported by 16M, but only to ensure the stability of 8M, the excess part of the stability cannot be guaranteed.

If you want to change the littlefs capacity, you have to call the**LittlefsFormat** and **LittlefsDeinit** firstly, and then call this function.

Syntax

```
int LittlefsInit(
        uint32 offsetBlock,
        LITTLEFS_CAPACITY_E fileSysCap,
        LITTLEFS_CAPACITY_E spiFlashCap
    );
```

Parameters

offsetBlock

[in]This parameter represents the index of the file system sector, and the one block is 4K. Such as littlefs starts from at 2M, offsetBlock equals 512 (2M/4K = 512).

fileSysCap

[in]This represents the littlefs capacity, please select the right capacity according to the size of the flash, this parameter must be less than or equals to **spiFlashCap.** See LITTLEFS_CAPACITY_E.

spiFlashCap

[in] The total capacity of SPI flash. See LITTLEFS_CAPACITY_E. Such as if the **spiFlashCap** equals 8M, **fileSysCap** must be less than or equals to 8, and if **fileSysCap** equals 8M, the value of the **offsetBlock** parameter ranges from 0 to 2047.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

2. LittlefsDeinit

This function de-initializes littlefs.

Syntax

int LittlefsDeinit(void)

Parameters

None.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

3. LittlefsCreateDir

This function creates a new directory. Multilevel directories have to be created one by one, not at one time.

Syntax

int LittlefsCreateDir(const char *path);

Parameters

[in]The name of directory you want to create .

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

4. LittlefsOpenDir

This function opens a directory. Once open a directory can be used with read to iterate over files.

Syntax

int LittlefsOpenDir(lfs_dir_t * dir, const char *path);

Parameters

dir

[out]Pointer to the directory struction that contains a direrctory information.

See lfs_dir_t.

path

[in]The path of the directory.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

5. LittlefsReadDir

This functin reads an entry in the directory. Fills out the info structure, based on the specified directory.

Syntax

```
int LittlefsReadDir(lfs_dir_t *cwd, struct lfs_info *fileInfo);
```

Parameters

cwd

[in]The directory which you want to read. See lfs_dir_t.

fileInfo

[out] Fills out the info structure, based on the specified directory. See lfs_info.

Return Value

1 indicates that the info has been read from directory or file. 0 indicates that no info has been read. Others indicates failure, To get extended error information, call GetLastError.

6. LittlefsCloseDir

This function close a directory. Releases any allocated resources.

Syntax

```
int LittlefsCloseDir(lfs_dir_t *cwd);
```

Parameters

cwd

[in]Close the current working directory.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

7. LittlefsOpenFile

This function opens or creates a file. When the file operation is completed, it should be closed in time. And the same file can not be opened continously, otherwise, there will be an exception.

Syntax

```
int LittlefsOpenFile(lfs_file_t *file, const char *path, int openFlags);
```

Parameters

file
[out] Pointer to the struction that contains a file information. See lfs_file_t.
path
[in] The path of the file.
openFlags
[in] The mode that the file is opened in is determined by the **openFlags**, which are values from the enum lfs_open_flags that are bitwise-ored together.
The following table shows the possible values:

Value	Description
LFS_O_RDONLY	Open a file as read only
LFS_O_WRONLY	Open a file as write only
LFS_O_RDWR	Open a file as read and write
LFS_O_CREAT	Create a file if it does not exist
LFS_O_EXCL	Fail if a file already exists
LFS_O_TRUNC	Truncate the existing file to zero size
LFS_O_APPEND	Move to end of file on every write

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

8. LittlefsReadFile

This function read data from file.

Syntax

```
int LittlefsReadFile(
        lfs_file_t *file,
        void *buffer,
        uint32 size,
```

```
                uint32 *readLen
        );
```

Parameters

file
[in]The file that will read data from it. See lfs_file_t.
buffer
[out] Pointer to the buffer that the data has been read will be stored in this buffer.
size
[in] The number you want to read ,unit is byte.
readLen
[out] Pointer to the number of bytes read.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

9. LittlefsWriteFile

This function writes data to file. The file will not actually be updated on the storage until either **LittlefsFlushFile** or **LittlefsCloseFile** is called.

Syntax

```
int LittlefsWriteFile(
                lfs_file_t *file,
                const void *buffer,
                uint32 size,
                uint32 *lenWritten
        );
```

Parameters

file
[in]The file that will write data into it. See lfs_file_t.
buffer
[in] Pointer to the buffer that the data will be written is stored in this buffer.
size
[in] Number of bytes to write to the file.
lenWritten
[out]The length of the data actually written to the file.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information,

call GetLastError.

10. LittlefsCloseFile

This function closes a file. Any pending writes are written out to storage as though sync had been called and releases any allocated resources.

Syntax

int LittlefsCloseFile(lfs_file_t *file);

Parameters

file
[in]The file that will be closed. See lfs_file_t.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

11. LittlefsGetFilePointer

This function gets the position of the file. Equivalent to **LittlefsSetFilePointer**(file, 0, LFS_SEEK_CUR, &postion).

Syntax

int LittlefsGetFilePointer(lfs_file_t *file);

Parameters

file
[in]Get postion from the specified file. See lfs_file_t.

Return Value

Returns the position of the file.

12. LittlefsSetFilePointer

Change the position of the file. The change in position is determined by the offset and whence flag.

Syntax

int LittlefsSetFilePointer(lfs_file_t *file, int offset, int whence, int *position);

Parameters

file

[in] The file which will be changed the position. See lfs_file_t.

offset

[in] The relative position of origin. A positive value for this parameter moves the file pointer forward in the file, and a negative value moves the file pointer backward. You cannot use a negative value to move back past beyond the beginning of a file.

whence

[in] the absolutely you will be seek. The following table shows possible values.

Value	Description
LFS_SEEK_SET	Seek relative to an absolute position
LFS_SEEK_CUR	Seek relative to the current file position
LFS_SEEK_END	Seek relative to the end of the file

position

[out] Return the new position of the file.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

13. LittlefsDelete

This function removes a file or directory. If removing a directory, the directory must be empty.

Syntax

int LittlefsDelete(char *path);

Parameters

path

[in]The path of the file or directory.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

14. LittlefsFlushFile

This function synchronizes a file on storage. Any pending writes are written out to storage.

Syntax

int LittlefsFlushFile(lfs_file_t *file)

Parameters

file
[in]The file that will synchronize on stroage. See lfs_file_t.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

15. LittlefsGetFileSize

This function gets the size of the file. Similar to LittlefsSetFilePointer (file, 0, LFS_SEEK_END, position);

Syntax

int LittlefsGetFileSize(lfs_file_t *file);

Parameters

file
[in]Get the size from the file. See lfs_file_t.

Return Value

Returns the size of the file.

16. LittlefsSetFileSize

This function truncates the size of the file to the specified size.

Syntax

int LittlefsSetFileSize(lfs_file_t *file, uint32 size);

Parameters

file
[in]Set size of the specified file. See lfs_file_t.
size
[in]The new size of the file will be set.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

17. LittlefsRename

This function is used to rename or move a file or directory. If the destination exists, it must match the source in type. If the destination is a directory, the directory must be empty.

Note: If power loss occurs, it is possible that the file or directory will exist in both the oldpath and newpath simultaneously after the next mount.

Syntax

```
int LittlefsRename(
    const char *oldname,
    const char *newname
    );
```

Parameters

oldname
[in]The old path of the file or directory.
newname
[in]The new path of the file or directory.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

18. LittlefsFormat

This function formats a block device with the littlefs

Syntax

```
int LittlefsFormat(void);
```

Parameters

None.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

19. LittlefsDeviceFreeSpace

This function gets the disk free space.

Syntax

int LittlefsDeviceFreeSpace(uint32 *pFreeSize);

Parameters

pFreeSize
[out] Pointer to a variable to receive the number of free bytes on the disk.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

20. LittlefsDeviceUsedSpace

This function gets the disk space that has been used.

Syntax

int LittlefsDeviceUsedSpace(uint32 *pUsedSize);

Parameters

pUsedSize
[out] Pointer to a variable to receive the number of used bytes on the disk.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

21. LittlefsGetType

This function gets the littlefs type.

Syntax

int LittlefsGetType(lfs_t *littlefs);

Parameters

littlefs
[out]Retrun the littlefs.

Return Value

Return zero always.

22. FileSysUseCap

This function get the information about how the littlefs and fatfs use spi flash.

Syntax

int FileSysUseCap(FS_INIT_INFO_T *lfsInfo, FS_INIT_INFO_T *fatInfo);

Parameters

lfsInfo
[out]how the littlefs uses the flash, if **lfsInfo** is NULL, don't get information. See FS_INIT_INFO_T.

fatInfo
[out] how the fatfs uses the flash, if fatInfo is NULL, don't get information. See FS_INIT_INFO_T.

Return Value

Return zero always.

23. LittlefsVersion

This function gets the littlefs software version.

Syntax

int LittlefsVersion(char *LfsVersion);

Parameters

LfsVersion
[out]Get the littlefs version.

Return Value

Zero indicates success, -1 indicates failure.

Smart Card API

24. SmartCardReset

This function reset the smart card.

Syntax

```
uint32 SmartCardReset(
                    uint32 Voltage,
                    uint8* pAtrInfo,
                    uint32    len
                    );
```

Parameters

Voltage

[in]Setting reset voltage. The following table shows the possible value.

Value	Description
0	Set voltage to 1.8V
1	Set voltage to 3.0V

pAtrInfo

[out]Pointer to the info of the ATR(answer to reset).

len

[in] The size of the buff used to save ATR info.

Return Value

Return to the actual size of the ATR info. Equal zero indicates failure.

25. SmartCardSendInstr

This function send the smart card instruction.

Syntax

```
uint32 SmartCardSendInstr(
    uint8* instr,
    uint32 instrLen,
    uint8* result,
    uint32 resultLen,
    uint32 waitTime
    );
```

Parameters

instr

[in]Pointer to the instruction that will be sent.

instrLen

[in]The length of the instruction.

result

[out]An pointer to the feedback result after the instruction had sent.

resultLen

[in]The size of the buff that used to save the result data.

waitTime

[in]The wait response time(unit: ms), after send the instruction. If less than or equal to 50ms, use the default value, the default value is 50ms.

Return Value

Return to the actual size of the result data. Equal zero indicates failure.

Telephony api

1. TelephonyDial

This function can be called to dial a phone. The status of a call, e.g. dialling, idle, connection etc, will be reported via message notification mode. See TEL_NOTIFY_ID_E.

The message notification mode is that app register NotifyCallback function, the platform will notify the app of various messages through NotifyCallback .

Syntax

int TelephonyDial(BYTE* NUM);

Parameters

NUM

[in]Pointer to the telephony number.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

2. TelephonyAnswer

This function is used to listen to the phone, when there is an incoming call.

Syntax

int TelephonyAnswer(VOID);

Parameters

None.

Return Value

Equal to zero indicates success, -1 indicates failure.

3. TelephonyHangup

This function is used to hang up the phone.

Syntax

int TelephonyHangup(VOID);

Parameters

None.

Return Value

Equal to zero indicates success, -1 indicates failure.

Sms api

1. SmsReadText

This function get the text mode sms. The message reading result will be reported via message notification mode. See SMS_NOTIFY_ID_E.

The message notification mode is that app register NotifyCallback function, the platform will notify the app of various messages through NotifyCallback .

Syntax

```
int SmsReadText(BYTE index, ATC_CHARACTER_SET_TYPE_E type);
```

Parameters

index

[in]The sms position, it starts from 0.

type

[in] the character set type. See SMS_CHARACTER_SET_TYPE_E.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

2. SmsReadPdu

This function get the pdu mode sms. The message reading result will be reported via message notification mode. See SMS_NOTIFY_ID_E.

The message notification mode is that app register NotifyCallback function, the platform will notify the app of various messages through NotifyCallback .

Syntax

```
int SmsReadPdu(BYTE index);
```

Parameters

index

[in]The sms position, it starts from 0.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

3. SmsSendText

This function send the text mode sms. The message sending result will be reported via message notification mode.

The message notification mode is that app register NotifyCallback function, the platform will notify the app of various messages through NotifyCallback .

Syntax

```
int SmsSendText(
    BYTE*    pNum,
    BYTE numLen,
    SMS_CHARACTER_SET_TYPE_E type,
    BYTE*    pMsg,
    BYTE msgLen
);
```

Parameters

pNum

[in]Pointer to a BYTE of the phone number.

numLen

[in]the phone number length.

type

[in]the character set type. See SMS_CHARACTER_SET_TYPE_E.

pMsg

[in]Pointer to BYTE of the sms content buff.

msgLen

[in]the text length.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

The following table show the other possible values.

Value	Description
ERROR_SMS_SIM_NOT_READY	The sim card is not ready.
MC_SMS_IS_SENDING	A previous sms is sending.

4. SmsSendPdu

This function send the pdu mode sms. The message sending result will be reported via message notification mode.

The message notification mode is that app register NotifyCallback function, the platform will notify the app of various messages through NotifyCallback .

Syntax

```
int SmsSendPdu(BYTE* pPduStr, DWORD len);
```

Parameters

pPduStr
[in]Pointer to a BYTE of the pdu buff.
len
[in]the length of pdu.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.
The following table show the other possible values.

Value	Description
ERROR_SMS_SIM_NOT_READY	The sim card is not ready.
MC_SMS_IS_SENDING	A previous sms is sending.

5. SmsDelete

This function delete a sms.

Syntax

```
int SmsDelete(BYTE index);
```

Parameters

index
[in]The identity of the SMS that will be deleted.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

Network api

1. NetworkGetState

This function gets current network status.

Syntax

int NetworkGetState(MERCURY_NETWORK_STATUS_T* pstatus);

Parameters

pstatus

[out]Pointer to the structure of the MERCURY_NETWORK_STATUS_T contains network status information.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

2. NetworkGetSignalQuality

This function get current network signal quality.

Syntax

int NetworkGetSignalQuality(BYTE* prssi, WORD* prxfull)

Parameters

prssi

[out]Pointer to a BYTE that the received signal strength indication. The following table shows the rssi possible values.

Value	Description
0	-113dBm or less
1	-111dBm
2...30	-109...-53dBm
31	-51dBm or greater
99	unknown or not detectable

prxfull

[out]Pointer to a WORD that the received Signal quality.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error

information, call GetLastError.

3. NetworkSetAPN

This function sets PDP context parameter values for a PDP context. If APN is null then use the default APN, if user and password are null then we think it is not need password.

Syntax

```
int NetworkSetAPN(BYTE id, BYTE* apn, BYTE* user,    BYTE* password);
```

Parameters

id
[in] The PDP context identify, the value starts from 1 and currently only support **PDP_ID0**. See PDP_ID_E.

apn
[in] Pointer to a BYTE of access point name, this depends your network operator. If set to NULL, use the default value of "CMNET".

user
[in] Pointer to a BYTE of protocol configuration option pap user name.

password
[in] Pointer to a BYTE of protocol configuration option pap user password.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

4. NetworkOpenPDP

This function open the identity of the PDP contexts. The result is reported by a notification message. The notification class is **NOTIFY_CLASS_SOCKET**, and notification id is **PDP_NOTIFY_ID_E**.

Syntax

```
int NetworkOpenPDP(BYTE id);
```

Parameters

id
[in] the identity of the PDP contexts will be activated, The default value is 1.

See PDP_ID_E.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

5. NetworkClosePDP

This function close the identity of the PDP contexts. The result is reported by a notification message. The notification class is **NOTIFY_CLASS_SOCKET**, and notification id is **PDP_NOTIFY_ID_E**.

Syntax

int NetworkClosePDP(BYTE id);

Parameters

id
[in] The identity of the PDP contexts will be deactivated, the default value is 1. See PDP_ID_E.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

6. SocketCreate

This function create a socket.

Syntax

TCPIP_SOCKET_T SocketCreate(SOCKET_TYPE_E type)

Parameters

type
[in]The type of the socket that will be created. See SOCKET_TYPE_E.

Return Value

If the call is successful , return to the newly created socket descriptor, else return to -1. See TCPIP_SOCKET_T.

7. SocketConnect

This function initiate socket connection. This is for TCP client to connect to

server.

The connection result is reported by a notification message. The notification class is **NOTIFY_CLASS_SOCKET**, and notification id is **NOTIFY_ID_SOCKET_CONNECT**, see SOCKET_NOTIFY_ID_E. The data size is 8 bytes reported by the message, and the first 4 bytes are socket id, the last 4 bytes are error code.

Syntax

```
int SocketConnect(
    TCPIP_SOCKET_T so,
    SOCKET_ADDR_S* addr_ptr,
    int addr_len);
```

Parameters

so

[in]The socket handle will to be connected.

addr_ptr

[in]The pointer to address that connect the target address. See SOCKET_ADDR_S.

addr_len

[in]The address length.

Return Value

Equal to zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

8. SocketSend

This function description socket send, It is applied to TCP.

Syntax

```
int SocketSend(
    TCPIP_SOCKET_T so,
    char* buf,
    int len);
```

Parameters

so

[in]The socket handle to be connected. See TCPIP_SOCKET_T.

buf

[in]The pointer to the character that the data will be sent.

len

[in]The length of buffer.

Return Value

If successful, it will return to the actual length of the data sent. -1 indicates failure. To get extended error information, call GetLastError.

9. SocketSendTo

This function description socket send, It is applied to UDP.

Syntax

```
int SocketSendTo(
    TCPIP_SOCKET_T so,
    char* buf,
    int len,
    SOCKET_ADDR_S* to );
```

Parameters

so

[in]The socket handle to be connected. See TCPIP_SOCKET_T.

buf

[in]The pointer to the character that the data will be sent.

len

[in]The length of buffer.

to

[in]The pointer to a structure that target to send the address. See SOCKET_ADDR_S.

Return Value

If successful, it will return to the actual length of the data sent. -1 indicates failure. To get extended error information, call GetLastError.

10. SocketRecv

This function is used to receive data, it is applied to TCP.

Syntax

```
int SocketRecv(
    TCPIP_SOCKET_T so,
    char* buf,
    int len);
```

Parameters

so

[in]The socket handle to be connected. See TCPIP_SOCKET_T.

buf

[out]The pointer to the character that the receive data cache address.

len

[in]The length of buffer.

Return Value

If it is successful, return to the actual length of the data received, equal to zero indicates socket connection is closed. -1 indicates failure. To get extended error information, call GetLastError.

11. SocketRecvFrom

This function is used to receive data, it is applied to UDP.

Syntax

```
int SocketRecvFrom(
    TCPIP_SOCKET_T so,
    char* buf,
    int len,
    SOCKET_ADDR_S* from );
```

Parameters

so

[in]The socket handle to be connected. See TCPIP_SOCKET_T.

buf

[out]The pointer to the character that the receive data cache address.

len

[in] The length of buffer.

from

[out]The data source address pointer. See SOCKET_ADDR_S.

Return Value

If it is successful, return to the actual length of the data received, -1 indicates failure. To get extended error information, call GetLastError.

12. SocketClose

This function that close the socket. The result is reported by a notification message. The notification class is **NOTIFY_CLASS_SOCKET**, and notification id is **NOTIFY_ID_SOCKET_FULLCLOSE**, see SOCKET_NOTIFY_ID_E. The data size is 8 bytes

reported by the message, and the first 4 bytes are socket id, the last 4 bytes are error code.

Syntax

int SocketClose(TCPIP_SOCKET_T so);

Parameters

so

[in]The socket handle to be connected. See TCPIP_SOCKET_T.

Return Value

Equal to zero indicates success, -1 indicates failure.

13. SocketErrNo

This function gets the error code of the last socket function call.

Syntax

int SocketErrNo(TCPIP_SOCKET_T so);

Parameters

so

[in]The socket handle.

Return Value

Return the error code. The socket error code list are following.

macro definition	describe
ENOBUFS	1, Socket cache no enough
ETIMEDOUT	2, Socket connection timeout
EISCONN	3, Socket is already connected.
EOPNOTSUPP	4, Socket operation no supported
ECONNABORTED	5, Socket connection abnormal abort.
EWOULDBLOCK	6, Socket operation no completed(no results).
ECONNREFUSED	7, Socket connection request denied.
ECONNRESET	8, Socket connection is forced off by remote.
ENOTCONN	9, Socket not connected.
EALREADY	10, Socket is disconnected.
EINVAL	11, Socket operation invalid
EMSGSIZE	12, Application cache length exceeds UDP packet transmission capability . Application cache length does not meet the UDP packet reception capability.
EPIPE	13, Socket connection disconnected
EDESTADDRREQ	14, Need data to send target address.

ESHUTDOWN	15, Socket transmission terminated.
ENOPROTOOPT	16, Socket option is not supported.
EHAVEOOB	17, socket receives OOB（Out-of-band) data.
ENOMEM	18, Insufficient memory for TCPIP or system.
EADDRNOTAVAIL	19, The remote address is invalid.
EADDRINUSE	20, Socket local address has been bound.
EAFNOSUPPORT	21, Unsupported protocol family.
EINPROGRESS	22, Socket is in the connecting state.
ELOWER	23, lower layer (IP) error
ENOTSOCK	24, Socket operation on non-socket. Includes sockets which closed while blocked. It said that the socket handle is invalid.
EIEIO	27, bad input/output on Old Macdonald's farm
ETOOMANYREFS	28, Too many references
EFAULT	29, Socket operation error.
ENETUNREACH	30, Network is unreachable

14. INetNtoA

This function converts the numeric IP address to the IP address string, which results in a network byte order.

Syntax

```
char* INetNtoA( TCPIP_IPADDR_T ipaddr );
```

Parameters

ipaddr

[in]The numeric IP address (in network order). See TCPIP_IPADDR_T.

Return Value

Pointer to null-terminated string containing dotted-decimal printable representation of input parameter 'in'.

15. INetAtoN

This function converts the string IP address to a numeric IP address, which results in a network byte order, it does not support DNS parsing.

Syntax

```
int INetAtoN(
    char* ip_str,
```

```
    TCPIP_IPADDR_T* ipaddr_ptr
    );
```

Parameters

ip_str
[in]Pointer to the IP address string.

ipaddr_ptr
[out]Pointer to the numeric IP address.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

16. NetworkGetHostByName

This function resolve the IP address of the specified DNS, and the result is returned from a message.

The message notification mode is that app register NotifyCallback function, the platform will notify the app of various messages through NotifyCallback .

If resolve DNS successful, 8 bytes of the data returned by the message, The first 4 bytes are request id, the last 4 bytes are IP address. If DNS parsing failed, message notification returned to request id.

Syntax

```
TCPIP_HOST_HANDLE NetworkGetHostByName(
    char* name_ptr,
    uint32    time_out
    );
```

Parameters

name_ptr
[in]The name(URL) of the host to resolve.

time_out
[in]The host-parsing time out value (unit: ms).

Return Value

Return request handle if successful. Equal to zero indicates failure. See TCPIP_HOST_HANDLE.

17. FlightModeSet

The function sets the flight mode switch.

Syntax

int FlightModeSet(BOOLEAN enable);

Parameters

enable
[in] Boolean that specifies the state of the flight mode. If TRUE, the flight mode is opened; otherwise, exit flight mode.

Return Value

Zero indicates success, -1 indicates failure.

18. GprsAttach

The function does GPRS attach request. The result will be reported by a notification message. The notification class is **NOTIFY_CLASS_SOCKET**, and notification id is **PDP_NOTIFY_ID_E**.

Syntax

int GprsAttach(void);

Parameters

None;

Return Value

Return zero always, user can view the network states by **NetworkGetState**.

19. GprsDetacth

The function does GPRS detach request. The result will be reported by a notification message. The notification class is **NOTIFY_CLASS_SOCKET**, and notification id is **PDP_NOTIFY_ID_E**.

Syntax

int GprsDetach(void);

Parameters

None.

Return Value

Return zero always, user can view the network states by **NetworkGetState**.

20. SocketGetOpt

This function retrieves a socket option.

Syntax

int SocketGetOpt(TCPIP_SOCKET_T so, SOCKET_OPT_E opt, uint32* pdata);

Parameters

so

[in]Socket handle.

opt

[in] Socket option for which the value is to be retrieved. The following table shows the possible value.

Value	Description
SOCKET_TXDATA = 0	get count of bytes in sb_snd(socket output buffer).
SOCKET_RXDATA = 1	get count of bytes in sb_rcv (socket input buffer).
SOCKET_MYADDR = 2	return my IP address
SOCKET_ACK = 3	peer acked bytes in current TCP connection

data

[out] Pointer to the buffer in which the value for the requested option is to be returned.

Return Value

Zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

21. SocketGetState

This function gets a socket status.

Syntax

int SocketGetState(TCPIP_SOCKET_T so, short* state_ptr);

Parameters

so

[in]Socket handle.

state_ptr

[out] Pointer to socket state value. Socket state value will be any combinations of STATE bit macros with SS_ type which defined in Socket state bits.

Return Value

Zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

22. Network_GetPDPStatus

This function gets PDP status.

Syntax

BOOLEAN Network_GetPDPStatus(void);

Parameters

None.

Return Value

FALSE indicates de-activated. TRUE indicates activated.

23. Network_BaseStationInfoGet

This function gets the main base station information and a number of adjacent cell base stations information. And It can obtain up to 6 nearby base station information at most.

Syntax

uint32 Network_BaseStationInfoGet(MERCURY_CELLS_INFO_T* cell);

Parameters

cell

[out]Pointer to a MERCURY_CELLS_INFO_T structure that Saves access to base station information. see MERCURY_CELLS_INFO_T.

Return Value

Zero indicates API run successfully. Others indicates run failure. The following table shows the possible values.

Value	Description
0	API run successfully, no error
0x300000	API run failure, the pointer is null
0x300001	API run failure, not right parameter
0x300002	API run failure, length is out of range

0x300003	API run failure, the operation that user write default parameter value to ME is failure

24. Network_NetAddrGet

This function is used to get the IP address, subnet mask, gateway and DNS. This function can be used on the premise that PDP is activated. If the parameter is set to NULL to indicate that the parameter value is not obtained.

Syntax

int Network_NetAddrGet(char *pszIp, char *pszMask, char *pszGateway, char *pszDns);

Parameters

pszIp
[out]Get the IP address.

pszMask
[out]Get the subnet mask.

pszGateway
[out]Get the gateway.

pszDns
[out] Get the DNS.

Return Value

Zero indicates success. -1 indicates failure.

25. Network_ForceCampon

This function is used to lock the specified base station.

Syntax

void Network_ForceCampon(uint16 arfcn);

Parameters

arfcn
[in]The absolute radio frequency channel number.

Return Value

None.

26. Network_CancelForceCampon

This function is used to unlock the specified base station.

Syntax

```
void Network_CancelForceCampon(void);
```

Parameters

None;

Return Value

None;

27. Network_SetGprsMassRetransmitParam

This function sets gprs penalty time and retransmit num.

Syntax

```
int Network_SetGprsMassRetransmitParam(
                    uint32 penalty_time,
                    uint8 retransmit_num
                );
```

Parameters

penalty_time

[in] The prohibit time, unit is second. When the cell is re selected, the original cell will be banned for the penalty_time.

retransmit_num

[in]When the retransmit numbers greater than the parameter, it will automatically reselect the cell that meets the requirements.

Return Value

Zero indicates success. Others indicates failure.

28. Network_SetAuthType

This function is used to configure network authentication mode. This function have to be called before NetworkSetAPN.

Syntax

int Network_SetAuthType(PCO_AUTH_TYPE_E type);

Parameters

type
[in]The network authenticatio mode. See PCO_AUTH_TYPE_E.

Return Value

Retrun zero always.

29. Network_SelectBand

This function configuses the frequency band.

Syntax

int Network_SelectBand(int userBand);

Parameters

userBand
[in]The band to be selected. The frequency bands to support, see Supported Network Band.

Return Value

Zero indicates success. Others indicates failure.

30. Network_PingRequest (@deprecated)

This function sends ping echo request. You can't make network connections when making ping requests.
NOTE: This function will be deprecated beginning with V2.2.0.

Syntax

```
uint16 Network_PingRequest(
        char * faddr_ptr,
        uint32 data_size,
        uint32 time_out,
        TCPIP_PING_CALLBACK_FPTR callback_fptr
        );
```

Parameters

faddr_ptr

[in] Ping test address, it can be an IP or URL. e.g. "172.16.14.136" or "www.google.com".

data_size

[in] Ping payload size, unit: byte, range: 0 , and max data length that tcpip buffer can hold, e.g. when tcpip buffer is 1536, the Rang_max is 1492. If input is out of range, it will be confined automatically. 0 means default payload size 64 bytes.

time_out

[in] Ping echo request time out value, unit: ms. 0 means default time out value 4 seconds.

callback_fptr

[in] Ping echo reply callback pointer. See TCPIP_PING_CALLBACK_FPTR.

Return Value

non-0 is ping request handle; 0 indicates failure.

31. Network_PingCancel

This function cancel and stop ping request.

Syntax

void Network_PingCancel(uint16 pingHandle);

Parameters

pingHandle

[in]The ping handle generated from **Network_ PingV4Request** or **Network_ PingV6Request**.

Return Value

None.

32. Network_GetTaPwr

This function queries GSM TA.

Syntax

uint8 Network_GetTaPwr(uint8* pTa, uint8* pPwr);

Parameters

pTa

[out] Timing advance of GSM. The TA value range is 0-63, representing a time rang of 0-233us, equivalent to 0-70 km; each TA increase of 1, representing the device from the base station increased by about 550m.

pPwr

[out]Power control value of current business channel transmit. The following is the relationship between power control level and nominal output power.

GSM 400 and GSM 900 and GSM 850	
Power control level	**Nominal Output power (dBm)**
0~2	39
3	37
4	35
5	33
6	31
7	29
8	27
9	25
10	23
11	21
12	19
13	17
14	15
15	13
16	11
17	9
18	7
19~31	5
DCS 1800	
Power control level	**Nominal Output power (dBm)**
29	36
30	34
31	32
0	30
1	28
2	26
3	24
4	22
5	20
6	18
7	16
8	14
9	12
10	10

11	8
12	6
13	4
14	2
15~28	0

Return Value

1 indicates success, 0 indicates failure.

33. Network_PingV4Request

This function sends ping IPV4 echo request. You can't make network connections when making ping requests.

Syntax

```
uint16 Network_PingV4Request(
                    char* faddr_ptr,
                    uint32 data_size,
                    uint32 time_out,
                    TCPIP_PING_CALLBACK_EX_FPTR callback_fptr
                    );
```

Parameters

faddr_ptr

[in]Ping test address, it can be an IP or URL.

data_size

[in] Ping payload size, unit: byte, range: 0 , and max data length that tcpip buffer can hold, e.g. when tcpip buffer is 1536, the Rang_max is 1492. If input is out of range, it will be confined automatically. 0 means default payload size 64 bytes.

time_out

[in]Ping echo request time out value, unit: ms. 0 means default time out value 4 seconds.

callback_fptr

[in]Ping echo reply callback pointer. See TCPIP_PING_CALLBACK_EX_FPTR.

Return Value

non-0 is ping request handle; 0 indicates failure.

34. Network_PingV6Request

This function sends ping IPV6 echo request. You can't make network connections when making ping requests.

Syntax

uint16 Network_PingV6Request(char* faddr_ptr,uint32 data_size,uint32 time_out,TCPIP_PING_CALLBACK_EX_FPTR callback_fptr)

Parameters

faddr_ptr

[in]Ping test address, it can be an IP or URL.

data_size

[in] Ping payload size, unit: byte, range: 0 , and max data length that tcpip buffer can hold, e.g. when tcpip buffer is 1536, the Rang_max is 1492. If input is out of range, it will be confined automatically. 0 means default payload size 64 bytes.

time_out

[in]Ping echo request time out value, unit: ms. 0 means default time out value 4 seconds.

callback_fptr

[in]Ping echo reply callback pointer. See TCPIP_PING_CALLBACK_EX_FPTR.

Return Value

non-0 is ping request handle; 0 indicates failure.

35. NetworkSetPdpType

This function sets pdp type.

Syntax

int NetworkSetPdpType(uint8 type)

Parameters

type

[in]The pdp Type. The following table shows the possible value.

Value	Description
IP_TYPE	0, only support IPV4
IPV6_TYPE	1, only support IPV6
IPV4V6_TYPE	2, support IPV4 and IPV6

Return Value

Return 0 always.

Example

It must be called in the following order, and cannot be changed.

```
NetworkSetPdpType(IPV6_TYPE);
NetworkSetAPN(1,"CMNET",NULL,NULL);
NetworkOpenPDP(PDP_ID0);
```

36. SocketSetOpt

This function sets socket options, such as you can set whether TCP is keepalive or not.

Syntax

```
int SocketSetOpt(TCPIP_SOCKET_T so, int opt, uint32* pdata);
```

Parameters

so

[in]The socket handle.

opt

[in]The option name. The following table shows the possible value.

Value	Description
SO_KEEPALIVE = 0x0008	keep connections alive

pdata

[in] The option value pointer.

Return Value

0 for success, -1 for failure .

Example

Set keepalive or not:

```
uint32 keepAliveFlag = 0/1; (0: not keepalive; 1: keepalive)
SocketSetOpt (so, SO_KEEPALIVE, &keepAliveFlag);
```

37. SocketConnectV6

This function is used for IPV6 to establish a connection to the specified socket.

This is for TCP client to connect to server.

The connection result is reported by a notification message. The notification class is **NOTIFY_CLASS_SOCKET**, and notification id is **NOTIFY_ID_SOCKET_CONNECT**, see SOCKET_NOTIFY_ID_E. The data size is 8 bytes reported by the message, and the first 4 bytes are socket id, the last 4 bytes are error code.

Syntax

```
int SocketConnectV6(
                    TCPIP_SOCKET_T so,
                    V6_SOCKET_ADDR_S* addr_ptr,
                    int addr_len
                );
```

Parameters

so

[in]The socket handle will to be connected.

addr_ptr

[in]The pointer to address that connect the target address. See V6_SOCKET_ADDR_S.

addr_len

[in]The address length.

Return Value

Equal to zero indicates success, -1 indicates failure.

38. SocketSendToV6

This function is used to send UDP packets thar are not connected under IPV6.

Syntax

```
int SocketSendToV6(
                TCPIP_SOCKET_T so,
                char* buf,
                int len,
                V6_SOCKET_ADDR_S* to
            );
```

Parameters

so

[in]The socket handle to be connected. See TCPIP_SOCKET_T.

buf

[in]The pointer to the character that the data will be sent.

len

[in]The length of data.

to

[in] The data destination address pointer. See V6_SOCKET_ADDR_S.

Return Value

If successful, it will return to the actual length of the data sent. -1 indicates failure.

39. SocketRecvFromV6

This function is used to receive data under IPV6, it is applied to UDP.

Syntax

```
int SocketRecvFromV6(
        TCPIP_SOCKET_T so,
        char* buf,
        int len,
        V6_SOCKET_ADDR_S* from
    );
```

Parameters

so

[in]The socket handle to be connected. See TCPIP_SOCKET_T.

buf

[out]The pointer to the character that the receive data cache address.

len

[in] The length of buffer.

from

[out]The data source address pointer. See V6_SOCKET_ADDR_S.

Return Value

If it is successful, return to the actual length of the data received, -1 indicates failure.

40. ETH_SocketCreate

This function creates a specificed socket by ethernet card.

For ethernet card, you need to specify SPI CS logical ID first, and set up the relevant GPIO pins(such as reset and interrupt) to control 9051 ethernet card. The function is used in the following order:

1. ETH_RegInterface
2. ETH_DhcpRequest
3. ETH_SocketCreate

Syntax

```
TCPIP_SOCKET_T    ETH_SocketCreate(SOCKET_TYPE_E type, uint32 netID);
```

Parameters

type

[in]The type of the socket that will be created. See SOCKET_TYPE_E.

netID

[in]The network interface ID. The netID is obtained through **ETH_RegInterface**.

Return Value

If the call is successful , return to the newly created socket descriptor, else return to -1. See TCPIP_SOCKET_T.

41. ETH_RegInterface

This function registers the network interface.

Syntax

```
uint32 ETH_RegInterface(
                        uint32 spiID,
                        uint32 gpioRst,
                        uint32 gpioInt,
                        uint8* mac,
                        uint32 ipv6Flag
                    );
```

Parameters

spiID

[in]The spi cs logical ID.

gpioRst

[in]The GPIO reset pin.

gpioInt

[in]The GPIO interruput pin.

mac

[in]The MAC address of network card, like {0x00, 0x1E, 0x75, 0xB9, 0xFD, 0xB0 }.

ipv6Flag

[in]If ipv6Flag is 0, it means that IPV6 is not enabled, non-zero value means that IPv6 is enabled and address prefix length is also indicated.

Return Value

Return the network ID.

42. ETH_DeRegInterface

This function logs out the network interface.

Syntax

```
void ETH_DeRegInterface(uint32 netid);
```

Parameters

netid
[in] The network interface ID. The netID is obtained through **ETH_RegInterface**.

Return Value

None.

43. ETH_DhcpRequest

This function can dynamically get IP address under IPv4.

In the IPv4 scenario, when the bearer registration is successful, there are two ways to acquire IP:

1. Obtained by DHCP: **ETH_DhcpRequest.**
2. Configure IP manually: **ETH_SetIpAddress.**

In the IPv6 scenario, TCPIP will spontaneously initiate RS messages to obtain IP without configuration.

Syntax

```
int ETH_DhcpRequest(
    TCPIP_DHCP_CALLBACK_FPTR callback_fptr,
    uint32 time_out,
    uint32 net_id
 );
```

Parameters

callback_fptr
[in]The callback function pointer. See TCPIP_DHCP_CALLBACK_FPTR.
time_out
[in] DHCP time out value (unit: ms).
net_id
[in]The network id.

Return Value

0 indicates send DHCP request OK ; non-0 indicates send DHCP request fail.

44. ETH_DhcpCancel

This function cancels DHCP request.

Syntax

```
void ETH_DhcpCancel(uint32 net_id);
```

Parameters

net_id
[in]The network interface ID.

Return Value

None.

45. ETH_SetIpAddress

This function can set IP manually.
In the IPv4 scenario, when the bearer registration is successful, there are two ways to acquire IP:

1. Obtained by DHCP: **ETH_DhcpRequest.**
2. Configure IP manually: **ETH_SetIpAddress.**

In the IPv6 scenario, TCPIP will spontaneously initiate RS messages to obtain IP without configuration.

Syntax

```
int ETH_SetIpAddress(
                const TCPIP_NETIF_IPADDR_T*    addr_ptr,
                uint32 net_id
            );
```

Parameters

addr_ptr
[in] The addresses configure pointer. See TCPIP_NETIF_IPADDR_T.
net_id
[in]The network interface ID.

Return Value

0 indicates success, other value indicates failure. The following show the possible

value:

Value	Description
TCPIP_ERROR_OK	0, Success.
TCPIP_ERROR_INVALPARAM	1, invalid parameter.
TCPIP_ERROR_INVALNETID	2, invalid net id
TCPIP_ERROR_MEMALLOC	3, memory alloc fail.
TCPIP_ERROR_LOGICAL	4, calling or running logical error
TCPIP_ERROR_TIMEOUT	5, time out

46. ETH_GetIpAddress

This function gets IP addresses on given net interface ID.

Syntax

```
BOOLEAN ETH_GetIpAddress(
        TCPIP_NETIF_IPADDR_T*   addr_ptr,
        uint32    net_id
    );
```

Parameters

addr_ptr
[out]The addresses pointer, returned IP addresses are in Big-Endian. See TCPIP_NETIF_IPADDR_T.
net_id
[in]The net interface ID.

Return Value

TRUE indicates success; FALSE indicates failure.

47. Network_SetDnsV6

This function sets the DNS. This interface is suitable for GPRS link bearer and other link bearer(DM9051).

Syntax

```
int Network_SetDnsV6(
            TCPIP_IPADDR6_T* dns,
            uint32    net_id
        );
```

Parameters

dns
[in]The DNS pointer. See TCPIP_IPADDR6_T.
net_id
[in]The net interface ID. If net_id is -1, The default is GPRS link.

Return Value

0 indicates success, other value indicates failure. The following show the possible value:

Value	Description
TCPIP_ERROR_OK	0, Success.
TCPIP_ERROR_INVALPARAM	1, invalid parameter.
TCPIP_ERROR_INVALNETID	2, invalid net id
TCPIP_ERROR_MEMALLOC	3, memory alloc fail.
TCPIP_ERROR_LOGICAL	4, calling or running logical error
TCPIP_ERROR_TIMEOUT	5, time out

48. Network_GetIpv6Address

This function get the IPv6 address information. This interface is suitable for GPRS link bearer and other link bearer(DM9051).

Syntax

```
BOOLEAN Network_GetIpv6Address(
                    TCPIP_IPADDR6_T* ip6addr_ptr,
                    TCPIP_IPADDR6_T* localaddr_ptr,
                    TCPIP_IPADDR6_T* dns_ptr,
                    uint32 net_id
                );
```

Parameters

ip6addr_ptr
[out]The IPv6 address pointer.
localaddr_ptr
[out]The local address pointer.
dns_ptr
[out]The primary DNS pointer.
net_id
[in]The net interface ID. If net_id is -1, The default is GPRS link.

Return Value

TRUE indicates success; FALSE indicates failure.

49. ETH_PingV4Request

This function sends ping IPV4 echo request by ethernet link bearer. You can't make network connections when making ping requests.

Syntax

```
uint16 ETH_PingV4Request(
        char* faddr_ptr,
        uint32 data_size,
        uint32 time_out,
        TCPIP_PING_CALLBACK_EX_FPTR     callback_fptr,
        uint32 netid
    );
```

Parameters

faddr_ptr

[in]Ping test address, it can be an IP or URL. e.g. "172.16.14.136" or "www.google.com".

data_size

[in] Ping payload size, unit: byte, range: 0 , and max data length that tcpip buffer can hold, e.g. when tcpip buffer is 1536, the Rang_max is 1492. If input is out of range, it will be confined automatically. 0 means default payload size 64 bytes.

time_out

[in]Ping echo request time out value, unit: ms. 0 means default time out value 4 seconds.

callback_fptr

[in]Ping echo reply callback pointer. See TCPIP_PING_CALLBACK_EX_FPTR.

netid

[in]The net interface ID.

Return Value

non-0 is ping request handle; 0 indicates failure.

50. ETH_PingV6Request

This function sends ping IPV6 echo request by other link bearer. You can't make network connections when making ping requests.

Syntax

```
uint16 ETH_PingV6Request(
        char* faddr_ptr,
        uint32 data_size,
```

```
                    uint32 time_out,
                    TCPIP_PING_CALLBACK_EX_FPTR callback_fptr,
                    uint32 netid
                );
```

Parameters

faddr_ptr
[in]Ping test address, it can be an IP or URL. e.g. "172.16.14.136" or "www.google.com".
data_size
[in] Ping payload size, unit: byte, range: 0 , and max data length that tcpip buffer can hold, e.g. when tcpip buffer is 1536, the Rang_max is 1492. If input is out of range, it will be confined automatically. 0 means default payload size 64 bytes.
time_out
[in]Ping echo request time out value, unit: ms. 0 means default time out value 4 seconds.
callback_fptr
[in]Ping echo reply callback pointer. See TCPIP_PING_CALLBACK_EX_FPTR.
netid
[in]The net interface ID.

Return Value

non-0 is ping request handle; 0 indicates failure.

51. SocketCreateV6

This function is used for IPV6 to create a socket.

Syntax

```
TCPIP_SOCKET_T    SocketCreateV6(SOCKET_TYPE_E type);
```

Parameters

type
[in] The type of the socket that will be created. See SOCKET_TYPE_E.

Return Value

If the call is successful , return to the newly created socket descriptor, else return to -1. See TCPIP_SOCKET_T.

52. SocketBind

This function associates a local address with a socket.

Syntax

```
int SocketBind(
        TCPIP_SOCKET_T so,
        struct sci_sockaddr* addr_ptr,
        int addr_len
    );
```

Parameters

so
[in] Descriptor identifying an unbound socket.
addr_ptr
[in] Address to assign to the socket from the sci_sockaddr structure.
addr_len
[in] Length of the value in the addr_ptr parameter.

Return Value

Zero indicates success; -1 indicates failure.

53. SocketListen

This function places a socket at a state where it is listening for an incoming connection.

Syntax

```
int SocketListen( TCPIP_SOCKET_T so, int backlog);
```

Parameters

so
[in] Descriptor identifying a bound, unconnected socket.
backlog
[in]Maximum connecting num from client backlog can be set maximum as SOMAXCONN, backlog means maximum socket numbers in connecting(TCP handshake), backlog doesn't include socket connected!

Return Value

Zero indicates success; -1 indicates failure.

54. SocketAccept

This function permits an incoming connection attempt on a socket. SocketListen should be called at first!

Syntax

```
TCPIP_SOCKET_T SocketAccept(
        TCPIP_SOCKET_T so,
        struct sci_sockaddr* addr_ptr,
        int* addr_len
    );
```

Parameters

so
[in] Descriptor identifying a socket that has been placed in a listening state with the SocketListen function. The connection is actually made with the socket that is returned by this function.
addr_ptr
[out] New connection address from client. See sci_sockaddr.
addr_len
[in] Address length, this value is in fact no use, set with 0.

Return Value

Success: socket of the new accepted connection; failure: -1

55. SocketSelect

This function determines the status of one or more sockets, waiting if necessary, to perform synchronous I/O.

Syntax

```
int SocketSelect(
        MERCURY_FD_SET_S* in,
        MERCURY_FD_SET_S* out,
        MERCURY_FD_SET_S* ex,
        long tv
    );
```

Parameters

in
[in] Socket list for read event watching. See MERCURY_FD_SET_S.
out

[in] Socket list for write event watching. See MERCURY_FD_SET_S.

ex

[in] Socket list for exception event watching. See MERCURY_FD_SET_S.

tv

[in] Watching time-out value, unit: 0.1s.

Return Value

Greater than zero indicates watching event occured; equal to zero watching event time-out; equal to -1 failure, socket is invalid.

56. FdClr

This function clear socket from socket list.

Syntax

void FdClr(TCPIP_SOCKET_T so, MERCURY_FD_SET_S* set);

Parameters

so

[in]The socket which will be cleared. See TCPIP_SOCKET_T.

set

[in]Socket list pointer. See MERCURY_FD_SET_S.

Return Value

None.

57. FdSet

This function adds socket to socket list.

Syntax

void FdSet(TCPIP_SOCKET_T so, MERCURY_FD_SET_S* set);

Parameters

so

[in]The socket which will be added. See TCPIP_SOCKET_T.

set

[in]Socket list pointer. See MERCURY_FD_SET_S.

Return Value

None.

58. FdIsSet

This function checks if socket in socket list.

Syntax

int FdIsSet(TCPIP_SOCKET_T so, MERCURY_FD_SET_S* set);

Parameters

so
[in]The socket. See TCPIP_SOCKET_T.
set
[in]Socket list pointer. See MERCURY_FD_SET_S.

Return Value

1 if *so* is a member of the set. Otherwise, zero.

59. FdZero

This function clears socket list.

Syntax

void FdZero(MERCURY_FD_SET_S* set);

Parameters

set
[in] Socket list pointer. See MERCURY_FD_SET_S.

Return Value

None.

Audio api

1. AudioSetChannel

This function set the audio channel mode.

Syntax

int AudioSetChannel(AUDIO_DEVICE_MODE_TYPE_E channel);

Parameters

channel
[in]The audio channel mode. See AUDIO_DEVICE_MODE_TYPE_E.

Return Value

Zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

2. AudioGetChannel

This function gets the current audio channel mode.

Syntax

AUDIO_DEVICE_MODE_TYPE_E AudioGetChannel();

Parameters

None.

Return Value

The current mode of the audio channel. See AUDIO_DEVICE_MODE_TYPE_E.

3. AudioSetVolume

This function is used to set volume level.

Syntax

int AudioSetVolume(DWORD volume);

Parameters

volume
[in]the volume value of the speaker. The range of the volume level is from 1 to 9.

See AUDIO_VOLUME_LEVEL_E.

Return Value

Zero indicates success, -1 indicates failure. To get extended error information, call GetLastError.

4. AudioGetVolume

This function is used to get volume level.

Syntax

```
DWORD AudioGetVolume( );
```

Parameters

None.

Return Value

Return the volume level value of the speaker.

5. AudioDtmfPlay

This function plays a dtmf tone.

Syntax

```
int AudioDtmfPlay(
    MERCURY_DTMF_TONE_ID_E tone,
    BYTE time
);
```

Parameters

tone

[in] Dtmf tone enumeration value. See MERCURY_DTMF_TONE_ID_E.

time

[in] Number of milliseconds of playing this tone, the unit is 100ms. If time equals to zero, DTMF will be played all the time until you call AudioDtmfAbort.

Return Value

Zero indicates success. -1 indicates failure.

6. AudioDtmfAbort

This function aborts the DTMF tone.

Syntax

void AudioDtmfAbort(void);

Parameters

None.

Return Value

None.

7. AudioDtmfVolume

This function is used to set the DTMF volume size.

Syntax

void AudioDtmfVolume(uint32 volume);

Parameters

volume

[in]The size of the volume you want to set. The range of the volume is 0 to 65535, the default value is 16384. When the value is equal to 0, it's mute.

Return Value

None.

8. AudioSingleTonePlay

This function is used to play single tone.

Syntax

int AudioSingleTonePlay(uint32 freq, BYTE time)

Parameters

freq

[in]Audio frequency, unit is HZ. You can set the frequency you want, it can start from 0.

time

[in] Number of milliseconds of playing this tone, the unit is 100ms. If time equals to zero, Tone will be played all the time until you call AudioSingleToneAbort.

Return Value

Zero indicates success. -1 indicates failure.

9. AudioSingleToneAbort

This function aborts the single tone.

Syntax

void AudioSingleToneAbort(void)

Parameters

None.

Return Value

None.

10. AudioSingleToneVolume

This function can set the single tone volume.

Syntax

void AudioSingleToneVolume(uint32 volume)

Parameters

volume

[in]The size of the volume you want to set. The range of the volume is 0 to 65535, the default value is 16384. When the value is equal to 0, it's mute.

Return Value

None.

FOTA

1. FOTA_Init

This function initializes device information .

Syntax

void FOTA_Init(void);

Parameters

None.

Return Value

None.

2. FOTA_ImgInfoSet

This function sets upgrade mode, it contains storage location, path and update type. This function is called after write data over.

Syntax

```
int FOTA_ImgInfoSet(
        IMG_STORE_E storeType,
        const uint8* name,
        IMG_UPDATE_E updataType
    );
```

Parameters

storeType

[in]Set the storage location type of the upgrade packet.

The following table shows the possible values.

Value	Description
STORE_INSIDE_FLASH	0, Storage inside flash. in this type, the name parameter ignored, can set to NULL.
STORE_FILE_SYSTEM	1, Storage in file system, in this type, the name parameter cannot be greater than 256 bytes.
STORE_INSIDE_FLASH_EXT	2, Storage inside ps flash. in this type, the name parameter ignored, can set to NULL. The size of the region is 992kbytes. But it can only use in release version that is not support audio fuction.

name

[in]The specific storage path.

updataType

[in]Set to when does the upgrade begin.

The following table shows the possible values.

Value	Description
IMG_UPDATA_NOW	0, Upgrade right now.
IMG_UPDATA_NEXT_BOOT	1, Upgrade after next boot.
IMG_UPDATA_NEVER	2, NO auto upgrade, user need to upgrade manually.

Return Value

1 indicates success. Zero indicates failure. To get extended error information, call GetLastError.

3. FOTA_WroteLenGet

The function gets the breakpoint position, which is always 32kbyte integer times.

Syntax

uint32 FOTA_WroteLenGet(void);

Parameters

None.

Return Value

Return to the address of the breakpoint.

4. FOTA_FlashWrite

This function uses to write to flash for FOTA.

Syntax

int FOTA_FlashWrite(uint32 addr, const uint8 * buf, uint32 write_len);

Parameters

addr

[in]Write the data start from the address. The value is in the range 0 through 512kbyte. The address must start from 0 or breakpoint, if address is not 0 or the breakpoint position, then there will be an exception.

buf

[in]Pointer to the buffer that the data will be written to flash.

write_len

[in]The actual length, in bytes, of the data that want to write into the flash. The written address begins with the high 512K bytes, so the sum of **addr** and **write_len** cannot greater than 512*1024.

Return Value

1 indicates success. Zero indicates failure. To get extended error information, call GetLastError.

5. FOTA_FlashRead

This function reads upgrade packet into the buffer.

Syntax

```
BOOLEAN FOTA_FlashRead(uint32 addr, uint8* buf, uint32 read_len);
```

Parameters

addr

[in]Read the data start from the address. The value is in the range 0 through 512kbyte.

buf

[out]Pointer to a buffer to storage the data.

read _len

[in]The length, in bytes, of the date that wants to read. The sum of **addr** and **read_len** cannot greater than 512*1024.

Return Value

1 indicates success. Zero indicates failure. To get extended error information, call GetLastError.

6. APP_FlashWrite

This function uses to write to flash for app. Before write data, you must call **APP_FlashErase** to erase the blocks that you want program in.

Syntax

```
BOOLEAN APP_FlashWrite(
    uint32 addr,
    const uint8 * buf,
    uint32 write_len
```

```
);
```

Parameters

addr

[in] Write the data start from the address. The value is in the range 0 through 1024kbyte.

buf

[in] Pointer to the buffer that the data will be written to flash.

write_len

[in] The actual length, in bytes, of the data that want to write into the flash. The sum of **addr** and **write_len** cannot greater than 1024k.

Return Value

TRUE indicates success. FALSE indicates failure.

7. APP_FlashErase

This function erases the flash.

Syntax

BOOLEAN APP_FlashErase(uint32 addr);

Parameters

addr

[in] Erase the data start from the address. The value is in the range 0 through 1024kbyte. And Address must be an integer multiple of 32k byte, and erase by 32k bytes.

Return Value

TRUE indicates success. FALSE indicates failure.

8. APP_FlashRead

The function is used to read the flash.

Syntax

BOOLEAN APP_FlashRead(uint32 addr, uint8* buf, uint32 read_len);

Parameters

addr

[in] Read the data start from the address. The value is in the range 0 through 1024kbyte.

buf

[out] Pointer to a buffer to storage the data.

read_len

[in] The actual length, in bytes, of the data that want to read from the flash. The sum of **addr** and **read_len** cannot greater than 1024k.

Return Value

TRUE indicates success. FALSE indicates failure.

9. FOTA_RawDataInfoSet

This function can set update image by raw data without filesystem.

Syntax

```
int FOTA_RawDataInfoSet(
            uint32 cs_id,
            uint32 imgAddr,
            uint32 imgLen,
            uint32 enable
        );
```

Parameters

cs_id

[in]The spi cs logical id. If it is equal to -1, the default value is 1.

imgAddr

[in]The numbers of offset sectors, the size of a sector is 4096Bytes.

e.g. imgAddr = 3, it means that image's first address starts at 3*4096 in SPI Flash.

So when an image is written to spi flash, it needs to be a multiple of 4096 from the first address of SPI flash.

imgLen

[in]The length of the image that yout want to upgrade.

enable

[in]Set whether to upgrade next boot.

Return Value

0 indicates success, others indicates failure.

10. FOTA_RawDataClear

This function clear fota update flag. When upgraded to a new version, you need to call this function to clear the upgrade flag.

Syntax

int FOTA_RawDataClear(void);

Parameters

None.

Return Value

0 indicates success, others indicates failure.

11. LOGO_FlashWrite

This function can write data into logo partition.

Syntax

BOOLEAN LOGO_FlashWrite(uint32 addr, const uint8 * buf,uint32 read_len);

Parameters

addr
[in]The address values range from 0 to 160*1024. And the sum of addr and read_len is less than 160*1024。
buf
[in]The buffer for data to be written.
read_len
[in]The length, in bytes, of data that will be written into logo partition.

Return Value

TRUE indicates success. FALSE indicates failure.

12. LOGO_FlashErase

This function erases the logo partition's data.

Syntax

BOOLEAN LOGO_FlashErase(uint32 addr);

Parameters

addr
[in]Erase data from this address, and erase 32K at a time, so the values of addr can't be greater than (160-32)*1024, and the addr must be exactly 32K times or equal to zero.

Return Value

TRUE indicates success, FALSE indicates failue.

13. LOGO_FlashRead

This function reads data from logo partition.

Syntax

```
BOOLEAN LOGO_FlashRead(uint32 addr, uint8* buf, uint32 read_len);
```

Parameters

addr
[in] Read the data start from the address. The value is in the range 0 through 160K bytes.
buf
[out] Pointer to a buffer to storage the data.
read_len
[in] The actual length, in bytes, of the data that want to read from the patition. The sum of **addr** and **read_len** cannot greater than 160k.

Return Value

TRUE indicates success, FALSE indicates failue.

14. APP_FlashWriteExt

This function uses to write backup message into the ps flash area of flash for FOTA. The external region size is 992kbytes. This interval can also be used for customer secondary applications, etc. But it can only use in release version that is not support audio fuction.

Syntax

```
BOOLEAN APP_FlashWriteExt(
    uint32 addr,
    const uint8 * buf,
    uint32 read_len
);
```

Parameters

addr

[in]Write the data start from the address. The value is in the range 0 through 992kbyte. And the sum of addr and read_len is less than 992*1024。

buf

[in]Pointer to the buffer that the data will be written to flash.

read_len

[in]The actual length, in bytes, of the data that want to write into the flash.

Return Value

1 indicates success. Zero indicates failure.

15. APP_FlashEraseExt

This function erases the flash.

Syntax

BOOLEAN APP_FlashEraseExt(uint32 addr);

Parameters

addr

[in] Erase the data start from the address. The value is in the range 0 through 1024kbyte. And Address must be an integer multiple of 32k byte, and erase by 32k bytes.

Return Value

TRUE indicates success. FALSE indicates failure.

16. APP_FlashReadExt

The function is used to read the flash.

Syntax

BOOLEAN APP_FlashReadExt(uint32 addr, uint8* buf, uint32 read_len);

Parameters

addr

[in] Read the data start from the address. The value is in the range 0 through 992kbyte.

buf

[out] Pointer to a buffer to storage the data.

read_len

[in] The actual length, in bytes, of the data that want to read from the flash. The sum of **addr** and **read_len** cannot greater than 992k.

Return Value

TRUE indicates success. FALSE indicates failure.

I2C

1. I2C_Init

This function will initialize the i2c controller, and return a handler for the i2c slave device.

Syntax

INT I2C_Init (I2C_DEV *dev);

Parameters

dev

[in]Pointer to I2C_DEV structure for i2c slave device.

Return Value

If success, return a handle, whose value is more than or equal to 0. -1 indicates failure. To get extended error information, call GetLastError.

2. I2C_Deinit

This function will close the i2c controller.

Syntax

INT I2C_Deinit (uint32 handle);

Parameters

handle

[in] The i2c slave device has gotten by calling **I2C_Init**.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

3. I2C_Read

This function is used to start a read transfer.

Syntax

INT I2C_Read (uint32 handle, uint8 *reg_addr, uint8 *buffer, uint32 bytes);

Parameters

handle
[in] An i2c slave device has gotten by calling I2C_Init.
reg_addr
[in] The buffer to store i2c slave device's internal register address.
buffer
[out]These buffers will store the data, read from i2c slave device.
bytes
[in] The number of reading data.

Return Value

If success, return the number of reading data. -1 indicates failure. To get extended error information, call GetLastError.

4. I2C_Write

This function is used to start a write transfer.

Syntax

INT I2C_Write (uint32 handle, uint8 *reg_addr, uint8 *buffer, uint32 bytes);

Parameters

handle
[in] An i2c slave device has gotten by calling I2C_Init.
reg_addr
[in] The buffer to store i2c slave device's internal register address.
buffer
[in] These buffers will store the data, write to i2c slave device.
bytes
[in] The number of writing data.

Return Value

If success, return the number of writing data. -1 indicates failure. To get extended error information, call GetLastError.

5. I2C_Ioctl

This function will set i2c controller or get some status.

Syntax

INT I2C_Ioctl (uint32 handle, uint32 cmd, uint32 *arg);

Parameters

handle

[in] An i2c slave device has gotten by calling I2C_Init.

cmd

[in] Command type is defined in I2C Command Type.

arg

[in] The command parameter.

Return Value

Zero indicates success. -1 indicates failure. To get extended error information, call GetLastError.

SSL

Note: Starting with V2.2.0, These interfaces will be deprecated.

1. mercury_ssl_establish

This function establishes a SSL connection.

Syntax

```
unsigned int* mercury_ssl_establish(
                const char *host,
                unsigned short port,
                const char *ca_crt,
                size_t ca_crt_len);
```

Parameters

host

[in] Specify the hostname(IP) of the SSL server.

port

[in] Specify the SSL port of SSL server.

ca_crt

[in] Specify the root certificate which is PEM format.

ca_crt_len

[in] Length of root certificate, in bytes.

Return Value

return SSL handle.

2. mercury_ssl_destroy

This function destroy the specific SSL connection.

Syntax

int mercury_ssl_destroy(unsigned int* handle);

Parameters

handle

[in] Handle of the specific connection.

Return Value

Zero indicates success. A value less than zero indicates failure.

3. mercury_ssl_write

This function write data into the specific SSL connection. The function will return immediately if the data which has been written into the specific SSL connection is equals to **len**.

Syntax

```
int mercury_ssl_write(
                unsigned int* handle,
                const char *buf,
                int len,
                int timeout_ms
            );
```

Parameters

handle

[in] A descriptor identifying a SSL connection.

buf

[in] A pointer to a buffer containing the data to be transmitted.

len

[in]The length, in bytes, of the data pointed to by the **buf** parameter.

timeout_ms

[in]Specify the timeout value in millisecond. In other words, the API block **timeout_ms** millisecond maximumly.

Return Value

Less than zero indicates the SSL connection error occur. Equals zero indicates that no any data be write into the SSL connection in **timeout_ms** timeout period. (0, len] indicates that the total number of bytes be written in **timeout_ms** timeout period.

4. mercury_ssl_read

This function reads data from the specific SSL connection with timeout parameter. The API will return immediately if the data which has been received from the specific SSL connection is equals to **len**.

Syntax

```
int mercury_ssl_read(
            unsigned int* handle,
            char *buf,
            int len,
            int timeout_ms
```

```
);
```

Parameters

handle
[in] A descriptor identifying a SSL connection.

buf
[out] A pointer to a buffer to receive incoming data.

len
[in] The length, in bytes, of the data pointed to by the **buf** parameter.

timeout_ms
[in] Specify the timeout value in millisecond. In other words, the API block **timeout_ms** millisecond maximumly.

Return Value

The following are possible values:

-2 : SSL connection error occur.
-1 : SSL connection be closed by remote server.
0 : No any data be received in **timeout_ms** timeout period.
(0, len]: The total number of bytes be received in **timeout_ms** timeout period.

TLS_SHA1

1. mercury_sha1_init

This function initializes SHA-1 context.

Syntax

```
void mercury_sha1_init( mercury_sha1_context *ctx );
```

Parameters

ctx
[in] SHA-1 context to be initialized. See mercury_sha1_context.

Return Value

None.

2. mercury_sha1_free

This function clears SHA-1 context.

Syntax

```
void mercury_sha1_free( mercury_sha1_context *ctx );
```

Parameters

ctx
[in] SHA-1 context to be cleared. See mercury_sha1_context.

Return Value

None.

3. mercury_sha1_clone

Clone (the state of) a SHA-1 context

Syntax

```
void mercury_sha1_clone(
    mercury_sha1_context *dst,
    const mercury_sha1_context *src
    );
```

Parameters

dst
[out] The destination context. See mercury_sha1_context.
src
[in] The context to be cloned. See mercury_sha1_context.

Return Value

None.

4. mercury_sha1_starts

SHA-1 context setup.

Syntax

```
void mercury_sha1_starts( mercury_sha1_context *ctx );
```

Parameters

ctx
[in]context to be initialized.

Return Value

None.

5. mercury_sha1_update

SHA-1 process buffer.

Syntax

```
void mercury_sha1_update(
        mercury_sha1_context *ctx,
        const unsigned char *input,
        size_t ilen
        );
```

Parameters

ctx
[in] SHA-1 context
input
[in] buffer holding the data
ilen
[in] length of the input data

Return Value

None.

6. mercury_sha1_finish

SHA-1 final digest.

Syntax

```
void mercury_sha1_finish(
        mercury_sha1_context *ctx,
        unsigned char output[20]
        );
```

Parameters

ctx
[in]SHA-1 context
output
[out]SHA-1 checksum result

Return Value

None.

7. mercury_sha1

Output = SHA-1(input buffer)

Syntax

```
void mercury_sha1(
        const unsigned char *input,
        size_t ilen,
        unsigned char output[20]
        );
```

Parameters

input
[in] buffer holding the data
ilen
[in] length of the input data
output
[out] SHA-1 checksum result

Return Value

None.

TLS_SHA256

1. mercury_ sha256_init

This function initializes SHA-256 context.

Syntax

```
void mercury_ sha256_init (mercury_sha256_context *ctx );
```

Parameters

ctx
[in] SHA-256 context to be initialized. See mercury_sha256_context.

Return Value

None.

2. mercury_sha256_free

This function clears SHA-256 context.

Syntax

```
void mercury_sha256_free( mercury_sha256_context *ctx );
```

Parameters

ctx
[in] SHA-256 context to be cleared. See mercury_sha256_context.

Return Value

None.

3. mercury_sha256_clone

Clone (the state of) a SHA-256 context

Syntax

```
void mercury_sha256_clone(
    mercury_sha256_context *dst,
    const mercury_sha256_context *src
    );
```

Parameters

dst
[out] The destination context. See mercury_sha256_context.
src
[in] The context to be cloned. See mercury_sha256_context.

Return Value

None.

4. mercury_sha256_starts

SHA-256 context setup.

Syntax

```
void mercury_sha256_starts( mercury_sha256_context *ctx, int is224 );
```

Parameters

ctx
[in]context to be initialized. See mercury_sha256_context.
is224
[in] 0 = use SHA256, 1 = use SHA224.

Return Value

None.

5. mercury_sha256_update

SHA-256 process buffer.

Syntax

```
void mercury_sha256_update (
        mercury_sha256_context *ctx,
        const unsigned char *input,
        size_t ilen
        );
```

Parameters

ctx
[in] SHA-256 context
input
[in] buffer holding the data
ilen
[in] length of the input data

Return Value

None.

6. mercury_sha256_finish

SHA-256 final digest.

Syntax

```
void mercury_sha256_finish(
        mercury_sha1_context *ctx,
        unsigned char output[32]
        );
```

Parameters

ctx
[in]SHA-256 context
output
[out]SHA-224/256 checksum result

Return Value

None.

7. mercury_sha256

Output = SHA-256(input buffer)

Syntax

```
void mercury_sha256(
        const unsigned char *input,
        size_t ilen,
        unsigned char output[32],
        int is224
        );
```

Parameters

input
[in] buffer holding the data
ilen
[in] length of the input data
output
[out] SHA-224/256 checksum result
is224
0 = use SHA256, 1 = use SHA224

Return Value

None.

Related Info

Type definition

1. typedef signed long INT32;
2. typedef unsigned int size_t;
3. typedef signed long LONG, *PLONG;
4. typedef unsigned int HANDLE;
5. typedef void * LPOVERLAPPED,LPVOID, HLOCAL, PVOID ,HWND;
6. typedef unsigned char uint8;
7. typedef unsigned short uint16;
8. typedef unsigned int uint32;
9. typedef unsigned long DWORD;
10. typedef int BOOL;
11. typedef int BOOLEAN;
12. typedef unsigned char BYTE;
13. typedef unsigned short WORD;

14. typedef float FLOAT;

15. typedef FLOAT *PFLOAT;

16. typedef BOOL *PBOOL;

17. typedef BOOL *LPBOOL;

18. typedef BYTE *PBYTE;

19. typedef BYTE *LPBYTE;

20. typedef int *PINT;

21. typedef int *LPINT;

22. typedef WORD *PWORD;

23. typedef WORD *LPWORD;

24. typedef long *LPLONG;

25. typedef DWORD *PDWORD;

26. typedef DWORD *LPDWORD;

27. typedef const void *LPCVOID;

28. typedef int INT;

29. typedef unsigned int UINT;

30. typedef unsigned int *PUINT;

31. typedef const char* LPTSTR;

32. typedef const unsigned short* LPCTSTR;

33. typedef void VOID;

34. typedef size_t SIZE_T;

35. typedef unsigned long* ULONG_PTR;

36. typedef unsigned long ULONG;

37. typedef void* HWND;

38. typedef void *TIMER_PTR;

39. typedef int TCPIP_SOCKET_T;

40. typedef unsigned int TCPIP_IPADDR_T;

41. typedef uint32 TCPIP_HOST_HANDLE;

implication

asynchronous gethostbyname request handle.

42. typedef uint16 TCPIP_PING_HANDLE;

Macro definition

1、 LMEM_FIXED

```
#define      LMEM_FIXED              0
```

2、 LMEM_ZEROINIT

```
#define      LMEM_ZEROINIT       0x0040
```

3、 LCD_RED/ LCD_GREEN /LCD_BLUE

```
#define LCD_RED      0xF800
#define LCD_GREEN    0x07E0
#define LCD_BLUE     0x001F
```

4、 MN_MAX_IMSI_ARR_LEN

```
#define MN_MAX_IMSI_ARR_LEN          8
```

5、 MNSIM_ICCID_ID_NUM_LEN

```
#define MNSIM_ICCID_ID_NUM_LEN      10
```

6、 TCPIP_IP6_ADDR_LEN_BYTES

```
#define TCPIP_IP6_ADDR_LEN_BYTES      16
```

1. Virtual Key Code definition

```
// customize
#define VK_OK          0x01
#define VK_SCAN        0x29
#define VK_ENTER       0x2b
#define VK_ESC         0x2c

// system
//If a key is used as both    VK_POWER and    VK_CANCEL, only use    VK_POWER.
```

```
// If a key is used as both    VK_POWER and    VK_CANCEL, only use    VK_POWER.
#define    VK_POWER        0x02
#define    VK_CANCEL       0x03
#define VK_BACK            0x08
#define VK_CLEAR           0x0C
#define VK_HOME            0x24
#define VK_LEFT            0x25
#define VK_UP              0x26
#define VK_RIGHT           0x27
#define VK_DOWN            0x28

// Define virtual key code.
/* VK_0 thru VK_9 are the same as ASCII '0' thru '9' (0x30 - 0x39) */
/* VK_A thru VK_Z are the same as ASCII 'A' thru 'Z' (0x41 - 0x5A) */

#define    VK_0            ('0')    // 0x30 ~ 0x39
#define    VK_1            ('1')
#define    VK_2            ('2')
#define    VK_3            ('3')
#define    VK_4            ('4')
#define    VK_5            ('5')
#define    VK_6            ('6')
#define    VK_7            ('7')
#define    VK_8            ('8')
#define    VK_9            ('9')
#define    VK_STAR         ('*')    // 0x2a
#define    VK_POUND        ('#')    // 0x23
#define    VK_AT           ('@')    // 0x40

#define VK_NUMPAD0         0x60
#define VK_NUMPAD1         0x61
#define VK_NUMPAD2         0x62
#define VK_NUMPAD3         0x63
#define VK_NUMPAD4         0x64
#define VK_NUMPAD5         0x65
#define VK_NUMPAD6         0x66
#define VK_NUMPAD7         0x67
#define VK_NUMPAD8         0x68
#define VK_NUMPAD9         0x69
#define VK_MULTIPLY        0x6A
#define VK_ADD             0x6B
#define VK_SEPARATOR       0x6C
#define VK_SUBTRACT        0x6D
#define VK_DECIMAL         0x6E
```

```
#define VK_DIVIDE          0x6F
#define VK_F1              0x70
#define VK_F2              0x71
#define VK_F3              0x72
#define VK_F4              0x73
#define VK_F5              0x74
#define VK_F6              0x75
#define VK_F7              0x76
#define VK_F8              0x77
#define VK_F9              0x78
#define VK_F10             0x79
#define VK_F11             0x7A
#define VK_F12             0x7B
#define VK_F13             0x7C
#define VK_F14             0x7D
#define VK_F15             0x7E
#define VK_F16             0x7F
#define VK_F17             0x80
#define VK_F18             0x81
#define VK_F19             0x82
#define VK_F20             0x83
#define VK_F21             0x84
#define VK_F22             0x85
#define VK_F23             0x86
#define VK_F24             0x87

#define VK_VOLUME_DOWN     0xAE
#define VK_VOLUME_UP       0xAF

/*for qwery keypad*/
#define   VK_Q             ('Q')
#define   VK_W             ('W')
#define   VK_E             ('E')
#define   VK_R             ('R')
#define   VK_T             ('T')
#define   VK_Y             ('Y')
#define   VK_U             ('U')
#define   VK_I             ('I')
#define   VK_O             ('O')
#define   VK_P             ('P')
#define   VK_A             ('A')
#define   VK_S             ('S')
#define   VK_D             ('D')
#define   VK_F             ('F')
```

```
#define VK_G        ('G')
#define VK_H        ('H')
#define VK_J        ('J')
#define VK_K        ('K')
#define VK_L        ('L')
#define VK_Z        ('Z')
#define VK_X        ('X')
#define VK_C        ('C')
#define VK_V        ('V')
#define VK_B        ('B')
#define VK_N        ('N')
#define VK_M        ('M')
```

2. Com Baud Rate

```
#define BAUD_1200       0x2A50
#define BAUD_2400       0x1528
#define BAUD_4800       0x0A94
#define BAUD_9600       0x054A
#define BAUD_19200      0x02A5
#define BAUD_38400      0x0152
#define BAUD_57600      0x00E2
#define BAUD_115200     0x0071
#define BAUD_230400     0x0038
#define BAUD_460800     0x001C
#define BAUD_921600     0x000E
#define BAUD_1625000    0x8
#define BAUD_3250000    0x4
```

3. Parity bits

```
#define DISABLE_PARITY  0
#define EVEN_PARITY     1
#define ODD_PARITY      2
```

4. Byte size

```
#define DEFAULT_BITS    0   // default is 8bit
#define FIVE_BITS       1
#define SIX_BITS        2
#define SEVEN_BITS      3
```

```
#define     EIGHT_BITS          4
```

5. flow control

```
#define     NO_FLOW_CONTROL     0
#define     HW_FLOW_CONTROL     1
#define     SW_FLOW_CONTROL     2
```

6. stop bit

```
#define     DEFAULT_STOP_BIT    0    // 1BIT
```

7. Socket state bits

```
#define SS_NOFDREF          0x0001   /* no file table ref any more */
#define SS_ISCONNECTED      0x0002   /* socket connected to a peer */
#define SS_ISCONNECTING     0x0004   /* in process of connecting to peer */
#define SS_ISDISCONNECTING  0x0008   /*  in process  of disconnecting */
#define SS_CANTSENDMORE     0x0010   /* can't send more data to peer */
#define SS_CANTRCVMORE      0x0020   /* can't receive more data from peer */
#define SS_RCVATMARK        0x0040   /* at mark on input */
#define SS_PRIV             0x0080   /* privileged for broadcast, raw... */
#define SS_NBIO             0x0100   /* non-blocking ops */
#define SS_ASYNC            0x0200   /* async i/o notify */
#define SS_UPCALLED         0x0400   /* zerocopy data has been upcalled (for select) */
#define SS_INUPCALL         0x0800   /* inside zerocopy upcall (reentry guard) */
#define SS_UPCFIN           0x1000   /* inside zerocopy upcall (reentry guard) */
#define SS_WASCONNECTING    0x2000   /* SS_ISCONNECTING w/possible pending
                                        error */
```

8. I2C Command Type

```
#define I2C_CTL_G_FREQ     0x20   /*get frequency*/
#define I2C_CTL_S_FREQ     0x21   /*set frequency*/
#define I2C_CTL_G_PORT     0x22   /*get port*/
#define I2C_CTL_S_PORT     0x23   /*set port*/
#define I2C_CTL_STOP_BUS 0x24   /*stop i2c bus*/
```

9. Supported Network Band

```
#define NET_BAND_GSM                    0x00
#define NET_BAND_DCS                    0x01
#define NET_BAND_GSM_DCS                0x02
#define NET_BAND_PCS                    0x03
#define NET_BAND_GSM850                 0x04
#define NET_BAND_GSM_PCS                0x05
#define NET_BAND_GSM850_DCS             0x06
#define NET_BAND_GSM850_PCS             0x07
#define NET_BAND_GSM850_GSM             0x08
#define NET_BAND_GSM850_GSM_PCS         0x09
#define NET_BAND_GSM850_GSM_DCS         0x0a
#define NET_BAND_GSM_DCS_PCS            0x0b
#define NET_BAND_GSM850_GSM_DCS_PCS     0x0c
#define NET_BAND_DCS_PCS                0x0d
#define NET_BAND_GSM850_DCS_PCS         0x0e
```

Callback Function definition

1. typedef VOID (*TIMER_FUN)(ULONG);

implication

The timer timeout callback function prototypes

2. SYMBOLHANDLECALLBACK

The callback function receives the scan feedback result. See SYMBOL_RESULT_T .

Syntax

```
typedef DWORD (*SYMBOLHANDLECALLBACK)(
    SYMBOL_RESULT_T *result
    );
```

3. NotifyCallback

This function is used to prototype a function to pass to RegNotifyCallback to call back the client when a registry value changes.

Syntax

```
typedef void (*NotifyCallback)(WORD,WORD ,void* ,DWORD);
```

implication

App registered the call back function, platform will notify app various messages through the NotifyCallback. That registered for notifications using RegNotifyCallback.

The first parameter is the receive message class, see NOTIFY_CLASS_E.

The second parameter is the operate result type, the following table shows the possible message id value.

Value	Description
PDP_NOTIFY_ID_E	The PDP operating results.
SMS_NOTIFY_ID_E	The SMS operating results
SOCKET_NOTIFY_ID_E	The SOCKET operating results
DNS_NOTIFY_ID_E	The DNS operating results
MC_CHR_NOTIFY_ID_E	Get the battery charge status results

TEL_NOTIFY_ID_E	The telephony result
MC_SCREEN_NOTIFY_ID_E	The screen state
MC_POWER_NOTIFY_ID_E	Power key state
MC_BARSCAN_NOTIFY_ID_E	Camera bar scan results
MC_TTS_NOTIFY_ID_E	Play text results
SYSTEM_NOTIFY_ID_E	SIM and PS off or on status

The third parameter is pointer to the structure of the specific data content, reference SMS_REC_TEXT_S , SMS_REC_PDU_S and so on.

The fourth parameter is the data size.

4. typedef void (*InterruptCallback)(void);

5. TCPIP_PING_CALLBACK_FPTR

This callback function pings result callback function.

Syntax

```
typedef void (*TCPIP_PING_CALLBACK_FPTR)(
                TCPIP_PING_RESULT_E     res,
                uint32                  time_delay,
                TCPIP_PING_HANDLE       ping_handle
);
```

implication

res

[in] Ping result, 0 indicates succeed; other indicates failure. The following shows the possible error values:

Value	Description
PINGRES_SUCCESS	0, ping OK, received ping echo reply
PINGRES_DNS_TIMEOUT	1, ping fail, DNS timeout
PINGRES_DNS_ERROR	2, ping fail, DNS error
PINGRES_ICMP_TIMEOUT	3, ping fail, icmp timeout
PINGRES_ICMP_ERROR	4, ping fail, icmp error

time_delay

[in] Ping time delay, only valid when success, unit: ms

ping_handle

[in] PING request handle. See TCPIP_PING_HANDLE;

6. MECURY_LOGO_UPDATA_CALLBACK_FPTR

This callback function is used to load logo image datafrom rom code or file.

Syntax

```
typedef int (*MECURY_LOGO_UPDATA_CALLBACK_FPTR)(
                    uint8 *pbuf,
                    uint32 rlen
                );
```

Parameters

pbuf
[out]The buffer to storage data which read from ROM code or file. Due to space constrains, it is recommended to store logo data in file system.
rlen
[in]The size of the data to be loaded at one time.

Return value

Length of the data actually read.

7. TCPIP_DHCP_CALLBACK_FPTR

This callback function obtains IP address result by DHCP.

Syntax

```
typedef void (*TCPIP_DHCP_CALLBACK_FPTR)(
            int res,
            const TCPIP_NETIF_IPADDR_T* addr_ptr,
            TCPIP_IPADDR_T dhcp_addr,
            uint32 netid );
```

implication

res
DHCP result - 0: OK; else - error. If get IP error by DHCP, you can set IP manually by calling **ETH_SetIpAddress** again.
addr_ptr
The ip addresses pointer. See TCPIP_NETIF_IPADDR_T.
dhcp_addr
The DHCP server address pointer.
netid
The net interface ID

8. TCPIP_PING_CALLBACK_EX_FPTR

This callback function pings result callback function.

Syntax

```
typedef void (*TCPIP_PING_CALLBACK_EX_FPTR)(
        TCPIP_PING_RESULT_E        res,
        uint32                     time_delay,
        uint16          ping_handle,
        uint8                      ttl,
        char*                      ipaddr);
```

implication

res

[in] Ping result, 0 indicates succeed; other indicates failure. The following shows the possible error values:

Value	Description
PINGRES_SUCCESS	0, ping OK, received ping echo reply
PINGRES_DNS_TIMEOUT	1, ping fail, DNS timeout
PINGRES_DNS_ERROR	2, ping fail, DNS error
PINGRES_ICMP_TIMEOUT	3, ping fail, icmp timeout
PINGRES_ICMP_ERROR	4, ping fail, icmp error

time_delay

[in] Ping time delay, only valid when success, unit: ms

ping_handle

[in] PING request handle.

ttl

[in]The Hop Limit .

ipaddr

[in]The ip address that you want to ping.

Structure definition

1. LPSECURITY_ATTRIBUTES

Syntax

```
typedefstruct _SECURITY_ATTRIBUTES {
DWORD nLength;
LPVOID lpSecurityDescriptor;
BOOL bInheritHandle ;
} SECURITY_ATTRIBUTES， * LPSECURITY_ATTRIBUTES;
```

Members

nLength

Struct size

lpSecurityDescriptor

Descriptor

bInheritHandle

New Object herit Old Object.

2. LPTHREAD_START_ROUTINE

Syntax

typedef VOID (*LPTHREAD_START_ROUTINE) (ULONG argc, LPVOID lpThreadParameter);

3. LPCRITICAL_SECTION

Syntax

```
typedef struct _RTL_CRITICAL_SECTION {
    LONG LockCount;
    LONG RecursionCount;
    HANDLE OwningThread;
```

```
        HANDLE LockSemaphore;
        ULONG_PTR SpinCount;
    } CRITICAL_SECTION, *LPCRITICAL_SECTION;
```

4. MERCURY_MESSAGE_S

This structure is the messages sent by the task.

Syntax

```
typedef struct
{
    unsigned short MessageID;
    unsigned int dataLen;
    void* pdata;
}MERCURY_MESSAGE_S;
```

Members

MessageID

The message identifier.

dataLen

The length of the message.

pdata

The message data.

5. TIMER_CONFIG_S

Syntax

```
typedef struct
{
    const char* timer_name;
    TIMER_FUN timer_fun;
    ULONG input;
    ULONG timer_expire;
    ULONG auto_activate;
    ULONG period;
}TIMER_CONFIG_S;
```

Members

timer_name

Timer name string, it can't be set to NULL.

timer_fun

Timeout callback function, it can't be set to NULL. See TIMER_FUN.

input

Input parameters for timeout callback function

timer_expire

Timer timeout (unit: ms), it can't be equal to 0.

auto_activate

Whether the timer is created and start timing:

SCI_NO_ACTIVATE: not effective, until user calls to ActiveTimer.

SCI_AUTO_ACTIVATE: auto effective

period

Whether it is a periodic timer.

6. SYSTEMTIME

This structure contains the system time information.

Syntax

```
typedef struct _SYSTEMTIME {
    WORD wYear; //2000 ~ 2049
    WORD wMonth; //1-12
    WORD wDayOfWeek;//0-6
    WORD wDay;//1-31
    WORD wHour;//0-23
    WORD wMinute;//0-59
    WORD wSecond;//0-59
    WORD wMilliseconds;//0-999
}SYSTEMTIME;
```

Members

wYear

Specifies the current year. [2000, 2049]

wMonth

Specifies the current month; 1 indicates January, 2 February, and so on.

wDayOfWeek

Specifies the current day of the week; Sunday = 0, Monday = 1, and so on.

wDay

Specifies the current day of the month.

wHour

Specifies the current hour - [0, 23].

wMinute

Specifies the current minute - [0,59].

wSecond

Specifies the current second - [0,59]

wMilliseconds

Specifies the current millisecond - [0,999].

7. POINT

This structure defines the x- and y-coordinates of a point.

Syntax

```
typedef struct _POINT {
    WORD left;
    WORD top;
} POINT, *PPOINT;
```

Members

left

Specifies the x-coordinate of the point.

top

Specifies the y-coordinate of the point.

8. RECTL

This structure defines the coordinates of the upper-left and lower-right corners of a rectangle.

Syntax

```
typedef struct _tagRECTL {
    WORD left;
    WORD top;
    WORD width;
    WORD height;
}RECTL, *PRECTL, *LPRECTL;
```

Members

`left`
Specifies the x-coordinate of the upper-left corner of the rectangle.

`top`
Specifies the y-coordinate of the upper-left corner of the rectangle.

width
Specifies the width of the rectangle.

height
Specifies the height of the rectangle.

9. DEVMODEW

This structure contains information about a printer environment and device initialization.

Syntax

```
typedef struct _devicemodew {
    DWORD dmBitsPerPel;
    DWORD dmPelsWidth;
    DWORD dmPelsHeight;
    DWORD dmDisplayFlags;
    DWORD dmDisplayFrequency;
    DWORD dmDisplayOrientation;
    DWORD dmDisplayTimeout;
    DWORD dmBacklightLevel;
} DEVMODEW,* PDEVMODEW,* NPDEVMODEW,* LPDEVMODEW;
```

Members

dmBitsPerPel

Specifies the color resolution, in bits per pixel, of the display device; for example, 4 bits for 16 colors, 8 bits for 256 colors, or 16 bits for 65,536 colors.

dmPelsWidth

Specifies the width, in pixels, of the visible device surface.

dmPelsHeight

Specifies the height, in pixels, of the visible device surface.

dmDisplayFlags

Specifies the device's display mode. set to zero.

dmDisplayFrequency

Specifies the frequency, in hertz (cycles per second), of the display device in a particular mode.

dmDisplayOrientation

Specifies the orientation of the screen. The default value is DMDO_0. The member is recorded with NV, data storage in power dump.

The following table shows the possible values.

Value	Description
DMDO_0	The screen is rotated by 0 degrees.
DMDO_90	The screen is rotated by 90 degrees counterclockwise.
DMDO_180	The screen is rotated by 180 degrees.
DMDO_270	The screen is rotated by 270 degrees counterclockwise.

dmDisplayTimeout

The screen brightness retentions timeout. The range of the timeout value is from 0 to 60s, if timeout equals to zero, the screen backlight keeps on. The default value is 30s. The member is recorded with NV, data storage in power dump.

dmBacklightLevel

The screen backlight brightness. The range of the backlight brightness value is from 0 to 0XE, the default value is 10. The member is recorded with NV, data storage in power dump.

10. COM_CONFIG_T

Syntax

```
typedef struct
{
    ULONG   baud_rate;
    BYTE    parity;
    BYTE    stop_bits;
    BYTE    byte_size;
    BYTE    flow_control;
} COM_CONFIG_T;
```

Members

baud_rate

UART transfer rate, used to explain the speed of data transmission. The specific value of baud rate refer to the macro definition: Com Baud Rate.

parity

Used to determine whether the received data bit error. Zero indicates no parity bits. See Parity bits.

stop_bits

At the end to mark the end of a character transfer. Zero indicates one stop bit. See stop bit.

byte_size

The size of the data bit. Zero indicates 8 data bits. See Byte size.

flow_control

For flow control mode. zero to represent without flow control, 1 to represent with flow control. See flow control.

11. SCAN_PARA_T

Syntax

```
typedef struct{
    SYMBOLHANDLECALLBACK    symbol_callback;
} SCAN_PARA_T;
```

Members

symbol_callback

the callback function is called for receive scan results. See SYMBOLHANDLECALLBACK.

12. SYMBOL_RESULT_T

The structure storage the camera scan result.

Syntax

```
typedef struct{
    SYMBOL_TYPE_T symbol_type;
    WORD    dataLen;
    unsigned char data[1];
} SYMBOL_RESULT_T;
```

Members

symbol_type

return the encoding format. See SYMBOL_TYPE_T.

dataLen

The length of the bar scanned actual data.

data[1]

Pointer to a string array for the bar scanned data to be processed. you can print and display these valid characters only. The valid length is dataLen.

13. SPI_CFG_S

This structure contains SPI configuration information.

Syntax

```
typedef struct
{
        char openFlag;
        SPI_MODE_E mode;
        uint32    tx_bit_length;
        uint32    freq;
} SPI_CONFIG_S;
```

Memberss

openFlag

No use(reserved);

mode

The spi mode. See SPI_MODE_E.

tx_bit_length

Transmit data bit number.

freq

SPI bus clock.

14. TTS_PARAM_INFO_S

This structure contains information about the TTS info.

Syntax

```
typedef struct
{
        int Volume; // MC_TTS_VOLUME_MIN~MC_TTS_VOLUME_MAX
        int ReadDigit; // read digit number ,see MC_TTS_READDIGIT_***
        int SpeakSpeed; // see MC_TTS_SPEED_***
        int Pitch; // see MC_TTS_PITCH_****
        int Channel; // see enumerations SOUND_CHANNEL_TYPE_E
}TTS_PARAM_INFO_S;
```

Members

Volume

The TTS play volume, the range of volume value is from -32768 to +32767,

and the default value is 0.

ReadDigit

The read digit number. the following table shows the possible values.

Value	Description
MC_TTS_READDIGIT_AUTO	0, default value, the device decides automatically
MC_TTS_READDIGIT_AS_NUMBER	1, say digit as number
MC_TTS_READDIGIT_AS_VALUE	2, say digit as value

SpeakSpeed

The TTS speak speed. the range of voice speed value is from -32768 to +32767, and the default value is 0.

Pitch

Set the TTS pitch. the range of voice tone value is from -32768 to +32767, and the default value is 0.

Channels

Which channel to be used to the TTS. The following table shows the possible vales.

Value	Description
SOUND_CHANNEL_HANDHOLD	0, handset mode
SOUND_CHANNEL_HANDFREE	1, Hands-Free mode
SOUND_CHANNEL_EARPHONE	2, headset mode
SOUND_CHANNEL_EARFREE	3, hands-free and headset mode

15. QR_ENC_CODE_T

This structure is used to receive the encode result.

Syntax

```
typedef struct {
    int width;
    int height;
    int rowSize;
    unsigned char* bits;
} QR_ENC_CODE_T;
```

Members

width

width of the QR code.

height
height of the QR code.
rowSize
row size of the QR code.
bits
pointer to the bit matrix data of the QR code. 1 means black, 0 means white.

16. SYSTEM_POWER_STATUS_EX2

This structure contains information about the power status of the system.

Syntax

```
typedef struct _SYSTEM_POWER_STATUS_EX2 {
    BYTE ACLineStatus;
    BYTE BatteryFlag;
    BYTE BatteryLifePercent;
    BYTE BatteryIsExist;
    DWORD BatteryLifeTime;
    DWORD BatteryFullLifeTime;
    BYTE Reserved2;
    BYTE BackupBatteryFlag;
    BYTE BackupBatteryLifePercent;
    BYTE Reserved3;
    DWORD BackupBatteryLifeTime;
    DWORD BackupBatteryFullLifeTime;
    DWORD BatteryVoltage;
    DWORD BatteryCurrent;
    DWORD BatteryAverageCurrent;
    DWORD BatteryAverageInterval;
    DWORD BatterymAHourConsumed;
    DWORD BatteryTemperature;
    DWORD BackupBatteryVoltage;
    BYTE BatteryChemistry;
    //  Add any extra information after the BatteryChemistry member.
}       SYSTEM_POWER_STATUS_EX2,        *PSYSTEM_POWER_STATUS_EX2,
*LPSYSTEM_POWER_STATUS_EX2;
```

Members

ACLineStatus
USB charge status. This member can be one of the values in the following table.

Value	Description

1	USB charging.
0	USB pull out

BatteryFlag

Battery charge status. This member can be a combination of the values in the following table. All other values are reserved.

Value	Description
BATTERY_FLAG_NORMAL	normal
BATTERY_FLAG_HIGH	High
BATTERY_FLAG_LOW	Low
BATTERY_FLAG_CRITICAL	Critical
BATTERY_FLAG_CHARGING	Charging
BATTERY_FLAG_UNKNOWN	Unknown status

BatteryLifePercent

Percentage of full battery charge remaining. This member can be a value in the range 0 to 100, or BATTERY_PERCENTAGE_UNKNOWN if the status is unknown.

BatteryIsExist

To judge that battery is exist. zero indicates battery is not exist, 1 indicates battery is exist.

BatteryLifeTime

Reserved; no used.

Number of seconds of battery life remaining, or BATTERY_LIFE_UNKNOWN if remaining seconds are unknown.

BatteryFullLifeTime

Reserved; no used.

Number of seconds of battery life when at full charge, or BATTERY_LIFE_UNKNOWN if full battery lifetime is unknown.

Reserved2

Reserved; set to zero.

BackupBatteryFlag

Reserved; no used.

Backup battery charge status. This member can be a combination of the following values:

- BATTERY_FLAG_HIGH
- BATTERY_FLAG_CRITICAL
- BATTERY_FLAG_CHARGING
- BATTERY_FLAG_NO_BATTERY
- BATTERY_FLAG_UNKNOWN
- BATTERY_FLAG_LOW

BackupBatteryLifePercent

Reserved; no used.

Percentage of full backup battery charge remaining. This value must be in the range 0 to 100, or BATTERY_PERCENTAGE_UNKNOWN.

Reserved3

Reserved; set to zero.

BackupBatteryLifeTime

Reserved; no used.

Number of seconds of backup battery life remaining, or BATTERY_LIFE_UNKNOWN if remaining seconds are unknown.

BackupBatteryFullLifeTime

Reserved; no used.

Number of seconds of backup battery life when at full charge, or BATTERY_LIFE_UNKNOWN if full battery lifetime is unknown.

BatteryVoltage

Amount of battery voltage in millivolts (mV).

BatteryCurrent

Reserved; no used.

Amount of instantaneous current drain in milliamperes (mA). This member can have a value in the range of 0 to 32,767 for charge, or 0 to –32,768 for discharge.

BatteryAverageCurrent

Reserved; no used.

Short-term average of device current drain (mA). This member can have a value in the range of 0 to 32,767 for charge, or 0 to –32,768 for discharge.

BatteryAverageInterval

Reserved; no used.

Time constant in milliseconds (ms) of integration used in reporting BatteryAverageCurrent.

BatterymAHourConsumed

Reserved; no used.

Long-term cumulative average discharge in milliamperes per hour (mAH). This member can have a value in the range of 0 to –32,768. This value can be reset by charging or changing the batteries.

BatteryTemperature

Battery temperature in degrees Celsius. This member can have a value in the range of –5 to 50; the increments are 1 degrees Celsius.

BackupBatteryVoltage

Reserved; no used.

Backup battery voltage in mV.

BatteryChemistry

Reserved; no used.

This can be one of the values in the following table.

Value	Description
BATTERY_CHEMISTRY_ALKALINE	Alkaline battery.
BATTERY_CHEMISTRY_NICD	Nickel Cadmium battery.
BATTERY_CHEMISTRY_NIMH	Nickel Metal Hydride battery.
BATTERY_CHEMISTRY_LION	Lithium Ion battery.
BATTERY_CHEMISTRY_LIPOLY	Lithium Polymer battery.
BATTERY_CHEMISTRY_ZINCAIR	Zinc Air battery.
BATTERY_CHEMISTRY_UNKNOWN	Battery chemistry is unknown.

17. NLED_COUNT_INFO

This structure contains information about the number of notification LEDs for the system.

Syntax

```
struct NLED_COUNT_INFO {
    UINT cLeds;
};
```

Members

cLeds
Count of LEDs for the system.

18. NLED_SUPPORTS_INFO

This structure contains information about the capabilities of the specified LED.

Syntax

```
struct NLED_SUPPORTS_INFO {
    UINT LedNum;
    LONG lCycleAdjust;
    BOOL fAdjustTotalCycleTime;
    BOOL fAdjustOnTime;
    BOOL fAdjustOffTime;
    BOOL fMetaCycleOn;
    BOOL fMetaCycleOff;
};
```

Members

LedNum
Number of the LED. The first LED is zero (0).

lCycleAdjust
Granularity of the cycle-time adjustments, in microseconds.

fAdjustTotalCycleTime
TRUE if the LED has an adjustable total cycle time; otherwise, it is FALSE.

fAdjustOnTime
TRUE if the LED has a separate on time; otherwise, it is FALSE.

fAdjustOffTime
TRUE if the LED has separate off time; otherwise, it is FALSE.

fMetaCycleOn
TRUE if the LED can blink n cycles, pause, and blink n cycles;otherwise, it is FALSE.

fMetaCycleOff
TRUE if the LED can blink n cycles, pause n cycles, and blink n cycles; otherwise, it is FALSE.

19. NLED_SETTINGS_INFO

This structure contains information about the capabilities of the specified LED.

Syntax

```
struct NLED_SETTINGS_INFO {
    UINT LedNum;
    INT OffOnBlink;
    LONG TotalCycleTime;
    LONG OnTime;
    LONG OffTime;
    INT MetaCycleOn;
    INT MetaCycleOff;
};
```

Members

LedNum

LED number. The first LED is zero (0).

OffOnBlink

Current setting. The following table shows the defined values.

Value	Description
0	Off
1	On
2	Blink

TotalCycleTime

Total cycle time of a blink, in microseconds.

OnTime

On time of the cycle, in microseconds.

OffTime

Off time of the cycle, in microseconds.

MetaCycleOn

Number of on blink cycles.

MetaCycleOff

Number of off blink cycles.

20. MERCURY_GPIO_CFG_S

This structure contains the interrupt information when set the interrupt mode.

Syntax

```
typedef struct
{
    uint16 default_val;
    MERCURY_DIR_E dir;
    MERCURY_INTERTUPT_MODE_E   int_sense;
    InterruptCallback    fun;
} MERCURY_GPIO_CFG_S;
```

Members

default_val

The default value of the GPIO level. The value is 0 or 1, and it is valid only in the OUTPUT mode.

dir

Set the GPIO state. See MERCURY_DIR_E.

int_sense

Set the interrupt trigger mode. See MERCURY_INTERTUPT_MODE_E.

fun

The callback will be call after trigger the interrupt. See InterruptCallback.

21. MCFILE_DATE_T

This structure contains the receives information about the found file.

Syntax

```
typedef struct
{
    uint8     mday;
    uint8     mon;
    uint16    year;
} MCFILE_DATE_T;
```

Members

mday

The day of the month. The value is in the range of 1 to 31.

mon

The months, the values is in the range 1 to 12.

year

The years, the value is in the range of 1980 to 2107.

22. MCFILE_TIME_T

This structure contains the receives information about the found file, Combine with MCFILE_DATE_T.

Syntax

```
typedef struct
{
    uint8     sec;
    uint8     min;
    uint8     hour;
}MCFILE_TIME_T;
```

Members

sec

The seconds after the minute, the value is in the range of 0 to 59.

min

The minutes after the hour, the value is in the range of 0 to 59.

hour

The hours since midnight, the value is in the range of 0 to 23.

23. MCFILE_FIND_DATA_TAG

This structure describes a file found by the FindFirstFile or the FindNextFile function.

Syntax

```
typedef struct MCFILE_FIND_DATA_TAG
{
    MCFILE_DATE_T    create_Date;
    MCFILE_TIME_T    create_time;
    MCFILE_DATE_T    modify_Date;
    MCFILE_TIME_T    modify_time;
    MCFILE_DATE_T    access_date;
    uint16           attr;
    uint32           length;
    uint16           name[MCFILE_MAX_PATH+1];
    uint8            short_name[13];
}MCFILE_FIND_DATA_T,*LPMCFILE_FIND_DATA_T;
```

Members

create_Date

This structure containing the date at which the file was created. See MCFILE_DATE_T.

create_time

This structure containing the time at which the file was created. See MCFILE_TIME_T.

modify_Date

This structure containing the date that the file was last written to. See MCFILE_DATE_T.

modify_time

This structure containing the time that the file was last written to. See MCFILE_TIME_T.

access_date

This structure containing the date at which the file was last accessed. See MCFILE_DATE_T.

attr

File attributes of the file found. The following table shows possible values. This member can be set to any combination of these values.

Value	Description
MCFILE_ATTR_FILE = 0x20	Indicates that the handle identifies a file.
MCFILE_ATTR_DIR = 0x30	Indicates that the handle identifies a

	directory.

length

The value of the file size, in bytes.

name

The name of the file.

short_name

An alternative name for the file. This name is in the classic 8.3 file name format. Reserved, user cannot use it.

24. SMARTCARD_EXTENSION

The structure is used by the driver to access all the other smart card data structures, and additional information. This structure is passed to all callback functions.

Syntax

```
typedef struct _SMARTCARD_EXTENSION {
    ULONG Version;
    VENDOR_ATTR VendorAttr;
    NTSTATUS (*ReaderFunction[16])(PSMARTCARD_EXTENSION);
    SCARD_CARD_CAPABILITIES CardCapabilities;
    ULONG LastError;
    struct {
        PULONG Information;
        PUCHAR RequestBuffer;
        ULONG RequestBufferLength;
        PUCHAR ReplyBuffer;
        ULONG ReplyBufferLength;
    } IoRequest;
    ULONG MajorIoControlCode;
    ULONG MinorIoControlCode;
    POS_DEP_DATA OsData;
    SCARD_READER_CAPABILITIES ReaderCapabilities;
    PREADER_EXTENSION ReaderExtension;
    SMARTCARD_REPLY SmartcardReply;
    SMARTCARD_REQUEST SmartcardRequest;
    T0_DATA T0;
    T1_DATA T1;
    ULONG Reserved[25];
} SMARTCARD_EXTENSION, *PSMARTCARD_EXTENSION;
```

Members

Version
The version of this structure.
VendorAttr
Mandatory vendor attribute data.
ReaderFunction
An array of smart card reader callback functions.
CardCapabilities
The capabilities of the currently inserted card.
LastError
The last error of an overlapped operation.
IoRequest
A structure containing the data of a user's I/O request.
MajorIoControlCode
The major I/O Control Code for the current request.
MinorIoControlCode
The minor I/O Control Code for the current request.
OsData
Pointer to an OS_DEP_DATA structure containing information.
ReaderCapabilities
Capabilities of the keyboard reader.
ReaderExtension
A pointer to reader specific data.
SmartcardReply
A buffer where the card reader stores all replies from the smart card.
SmartcardRequest
The current command to send to the smart card.
T0
Data for T=0.
T1
Data for T=1.
Reserved[25]
A buffer of 25 ULONG values, reserved for future use. Drivers should not use this space.

25. OS_DEP_DATA

This structure contains content that is dependent on the driver type.

Syntax

```
typedef struct _OS_DEP_DATA {
    struct _SMARTCARD_EXTENSION* pSmartCardExtension;
    CRITICAL_SECTION CritSect;
    HANDLE hChangeEvent;
```

```
    HANDLE hCancelEvent;
} OS_DEP_DATA, *POS_DEP_DATA;
```

Members

pSmartCardExtension
Pointer to a SMARTCARD_EXTENSION structure.

CritSect
Used to control entry into driver

hChangeEvent
Signaled on card-insertion event

hCancelEvent
Signaled on IOCTL_SMARTCARD_CANCEL_BLOCKING IOCTLs.

26. SCARD_CARD_CAPABILITIES

The structure holds all information about the currently inserted smart card.

Syntax

```
typedef struct _SCARD_CARD_CAPABILITIES {
    BOOLEAN InversConvention;
    ULONG etu;
    struct {
        UCHAR Buffer[64];
        UCHAR Length;
    } ATR;
    struct {
        UCHAR Buffer[16];
        UCHAR Length;
    } HistoricalChars;
    PCLOCK_RATE_CONVERSION ClockRateConversion;
    PBIT_RATE_ADJUSTMENT BitRateAdjustment;
    UCHAR Fl;
    UCHAR Dl;
    UCHAR II;
    UCHAR P;
    UCHAR N;
    ULONG GT;
    struct {
        ULONG Supported;
        ULONG Selected;
    } Protocol;
    struct {
```

```
        UCHAR WI;
        ULONG WT;
    } T0;
    struct {
        UCHAR IFSC;
        UCHAR CWI;
        UCHAR EDC;
        ULONG CWT;
        ULONG BWT;
        ULONG BGT;
    } T1;
    ULONG Reserved[25];
} SCARD_CARD_CAPABILITIES, *PSCARD_CARD_CAPABILITIES;
```

Members

InversConvention
TRUE indicates that the smart card uses the inverse convention.
etu
The calculated etu value for the smart card.
ATR.Buffer
The Answer-to-Reset string, after a warm or cold reset.
ATR.Length
The length of the ATR.
HistoricalChars.Buffer
A 16 byte buffer of historical data.
HistoricalChars.Length
The actual number of bytes used in HistoricalChars.Buffer.
PCLOCK_RATE_CONVERSION
A pointer to a clock rate conversion table.
PBIT_RATE_ADJUSTMENT
A pointer to a bit rate adjustment table.
FI
The clock rate conversion.
DI
The bit rate adjustment.
II
The smart card's maximum programming current.
P
The programming voltage, in units of 0.1 volts.
N
The amount of extra guard time in etu.
GT
The guard time in microseconds, including any extra guard time, for the minimum delay between two consecutive characters.

Protocol.Supported
A bit mask of the supported protocols.
Protocol.Selected
The currently selected protocol.
T0.WI
The T=0 waiting integer.
T0.WT
The T=0 work waiting time, in microseconds. This is the maximum delay between two consecutive characters.
T1.IFSC
The information field size of the smart card.
T1.CWI
The T=1 character waiting integer.
T1.BWI
The T=1 block waiting integer.
T1.EDC
The T=1 error detection code.
T1.CWT
The T=1 character waiting time, in microseconds. This is the maximum delay between two consecutive characters.
T1.BWT
The T=1 block waiting time, in microseconds. This is the maximum delay between the end of a block and the start of the next block sent in the opposite direction.
T1.BGT
The T=1 block guarding time, in microseconds. This is the minimum delay between the end of a block and the start of the next block sent in the opposite direction.
Reserved
An array of 25 ULONG values, reserved for future use. Smart card drivers should not use this space.

27. MERCURY_NETWORK_STATUS_T

The brief structure of signal, this signal indicate phone status.

Syntax

```
typedef struct
{
    MERCURY_PHONE_PLMN_STATUS_E    plmn_status;
    BOOL                           plmn_is_roaming;
    WORD                           mcc;
    WORD                           mnc;
```

```
    WORD                    mnc_digit_num;
    WORD                    lac;
    WORD                    cell_id;
}MERCURY_NETWORK_STATUS_T;
```

Members

plmn_status
the plmn statue. See MERCURY_PHONE_PLMN_STATUS_E.

plmn_is_roaming
if plmn is roaming plmn

mcc
the mcc

mnc
the mnc

mnc_digit_num
the mnc digit number

lac
the location area code

cell_id
the cell identity

28. SOCKET_ADDR_S

This structure contain the socket information.

Syntax

```
typedef struct
{
    unsigned short    port;
    unsigned long     ip_addr;
}SOCKET_ADDR_S;
```

Members

port
the port number.

ip_addr
the IP address.

29. SMS_REC_TEXT_S

This structure is text mode sms parsing structure. if The fourth parameter, in

the NotifyCallback function, equals to the size of structure(sizeof(SMS_REC_TEXT_S)) , it means read SMS success.

Syntax

```
typedef struct
{
        short index;
        short phoneNumLen;
        char    phoneNum[28];
        short msgLen;
        char      msg[500];
}SMS_REC_TEXT_S;
```

Members

index
the sms postion.

phoneNumLen
the phone number length.

phoneNum[28]
the phone number array.

msgLen
the message length.

msg[500]
the message content

30. SMS_REC_PDU_S

This structure is pdu mode sms parsing structure. if The fourth parameter, in the NotifyCallback function, equals to the size of structure(sizeof(SMS_REC_TEXT_S)) , it means read SMS success.

Syntax

```
typedef struct
{
        short index;
        short msgLen;
        char      msg[500];
}SMS_REC_PDU_S;
```

Members

index
the sms postion.

msgLen
the message length.

msg[500]
the message content.

31. SIM_IMSI_T

The structure contains the imsi information.

Syntax

```
typedef struct
{
    BYTE imsi_len;
    BYTE imsi_val[20];
} SIM_IMSI_T;
```

Members

imsi_len
The lenght of imsi(unit: byte).

imsi_val
The IMSI number of the SIM card, which is a ASCII code string.

32. SIM_ICCID_T

The structure contains the ccid information.

Syntax

```
typedef struct
{
    BYTE id_num[2*MNSIM_ICCID_ID_NUM_LEN + 1];
} SIM_ICCID_T;
```

Members

id_num
The ICCID number of the SIM card, which is a ASCII code string.

33. I2C_DEV

This structure contains the I2C initialization parameters.

Syntax

```
typedef struct
{
        uint32 id;
        uint32 freq;
        uint8 slave_addr;
        uint8 reg_addr_num;

} I2C_DEV;
```

Members

id

Logic id, which presents as a specific i2c bus and the i2c slave device connects to this bus.

freq

I2C slave device's working frequency.

slave_addr

I2C slave device's write address, whose length is 8 bits, not 7bits.

reg_addr_num

I2C slave device's internal register length.

34. MERCURY_CELLS_INFO_T

This structure uses to save the acquired base stations information.

Syntax

```
typedef struct
{
        NCELLS_INFO_T ncells[6];
        SCELL_INFO_T    scell;
}MERCURY_CELLS_INFO_T;
```

Members

ncells

Saves the acquired adjacent base station information pointer. See NCELLS_INFO_T.

scell

Save the acquired primary base station information pointer. See

SCELL_INFO_T.

35. NCELLS_INFO_T

This structure uses to save the acquired adjacent base stations information.

Syntax

```
typedef struct
{
    uint32 cell_exist;
    uint16  arfcn;
    uint8   bsic;
    uint8   rxlev;
    uint16  mcc;
    uint16  mnc;
    uint16  mnc_digit_num;
    uint16  lac;
    uint16  cell_id;
} NCELLS_INFO_T;
```

Members

Name	Meaning
cell_exist	Judge whether to obtain cell information. 1 indicates access to the corresponding base station information.
arfcn	absolute radio frequency channel number.
bsic	Base Station Identity Code.
rxlev	received signal level
mcc	mobile country code
mnc	mobile network code
mnc_digit_num	number of mnc
lac	location area code
cell_id	cell identity

36. SCELL_INFO_T

This structure uses to save the acquired primary base stations information.

Syntax

```
typedef struct
{
    uint32 cell_exist;
    uint16   arfcn;
    uint8    bsic;
    uint8    rxlev;
    uint16   mcc;
    uint16   mnc;
    uint16   mnc_digit_num;
    uint16   lac;
    uint16   cell_id;
} SCELL_INFO_T;
```

Members

Name	Meaning
cell_exist	Judge whether to obtain cell information. 1 indicates access to the corresponding base station information.
arfcn	absolute radio frequency channel number.
bsic	Base Station Identity Code.
rxlev	received signal level
mcc	mobile country code
mnc	mobile network code
mnc_digit_num	number of mnc
lac	location area code
cell_id	cell identity

37. TTS_PARAM_S

This structure defines the speak speed and pitch of TTS.

Syntax

```
typedef struct
{
    int SpeakSpeed;
    int Pitch;
}TTS_PARAM_S;
```

Members

SpeakSpeed

The speak speed of TTS, and the range of voice speed value is from -32768 to +32767, the default value is 0.

Pitch

The voice tone of TTS, and the range of voice tone value is from -32768 to +32767, the default value is 0.

38. lfs_dir_t

This structure defines the directory information about metedata.

Syntax

```
typedef struct lfs_dir {
    struct lfs_dir *next;
    uint16 id;
    uint8 type;
    lfs_mdir_t m;
    lfs_off_t pos;
    lfs_block_t head[2];
} lfs_dir_t;
```

Members

id

The entry id in the directory.

type

The file type. Such as LFS_TYPE_DIR(2)

m

The entry information strcucture of the directory. See lfs_mdir_t.

pos

The metadata of directory offset but contains with special offset for '.' and '..'.

head

directory head pairs.

39. lfs_mdir_t

This structure records information about the directory.

Syntax

```
typedef struct lfs_mdir {
    lfs_block_t pair[2];
    uint32 rev;
```

```
    lfs_off_t off;
    uint32 etag;
    uint16 count;
    BOOL erased;
    BOOL split;
    lfs_block_t tail[2];
} lfs_mdir_t;
```

Members

pair
A metadata pair is stored in two blocks, with one block acting as a redundant backup in case the other is corrupted. These two blocks could be anywhere in the disk and may not be next to each other.

rev
Incremented every update, only the uncorrupted metadata-block with the most recent revision count contains the valid metadata. Comparison between revision counts must use sequence comparison because therevision counts may overflow.

off
The metadata of directory offset.

etag
operations on 32-bit entry tags

count
the number of blocks by directory.

erased
Marks the erase status of the directory.

split
directory path split status.

tail
Pointer to the next metadata-pair in the filesystem. A null pair-pointer (0xffffffff, 0xffffffff) indicates the end of the list. If the highest bit in the dir size is set, this points to the next metadata-pair in the current directory. Otherwise, it points to an arbitrary metadata-pair. Starting with the superblock, the tail-pointers form a linked-list containing all metadata-pairs in the filesystem.

40. lfs_info

The file info structure.

Syntax

```
struct lfs_info {
    // Type of the file, either LFS_TYPE_REG or LFS_TYPE_DIR
```

```
        uint8 type;
        // Size of the file, only valid for REG files
        lfs_size_t size;
        // Name of the file stored as a null-terminated string
        char name[LFS_NAME_MAX+1];
};
```

41. lfs_file_t

This struction lists file property.

Syntax

```
typedef struct lfs_file {
        struct lfs_file *next;
        uint16 id;
        uint8 type;
        lfs_mdir_t m;

        struct lfs_ctz {
                lfs_block_t head;
                lfs_size_t size;
        } ctz;

        uint32 flags;
        lfs_off_t pos;
        lfs_block_t block;
        lfs_off_t off;
        lfs_cache_t cache;

        const struct lfs_file_config *cfg;
} lfs_file_t;
```

42. MERCURY_BOOT_IMAGE_S

This struction lists logo picture property.

Syntax

```
typedef struct
{
        uint32 magicNum;
        uint16 left;
```

```
        uint16 top;
        uint16 width;
        uint16 height;
        uint32 imageLen;
        uint32 lcdTypeIndex;
    }MERCURY_BOOT_IMAGE_S;
```

Members

magicNum:
The magic number is fixed value: 0x5a5aa5a5 , it cannot be changed.
left:
Specifies the x-coordinate of the starting point.
top:
Specifies the y-coordinate of the starting point.
width:
Width of the logo.
height:
Height of the logo.
imageLen:
Size of the logo picture. (width x height x 2)
lcdTypeIndex:
The type index of the LCD screen. The following table shows the possible values.

LCD Type	Description
ST7789H2_3WIRE_9BIT_2DATA	0
ST7789H2_4WIRE_8BIT_1DATA	1
ST7789V2_3WIRE_9BIT_2DATA	2
ILI9342C_4WIRE_8BIT_1DATA	3
ST7735_4WIRE_8BIT_1DATA	4

43. mercury_sha1_context

SHA-1 context structure.

Syntax

```
    typedef struct
    {
        unsigned int total[2];
        unsigned int state[5];
        unsigned char buffer[64];
    }mercury_sha1_context;
```

Members

total
number of bytes processed
state
intermediate digest state
buffer
data block being processed

44. mercury_sha256_context

SHA-256 context structure

Syntax

```
typedef struct
{
        unsigned int total[2];
        unsigned int state[8];
        unsigned char buffer[64];
        int is224;
}mercury_sha256_context;
```

Members

total
number of bytes processed
state
intermediate digest state
buffer
data block being processed
is224
0 => SHA-256, else SHA-224

45. V6_SOCKET_ADDR_S

This structure contain the socket connect information for IPV6.

Syntax

```
typedef struct
{
        unsigned short    port;              /* port number */
        unsigned char     ip_addr[16];       /* ip address */
}V6_SOCKET_ADDR_S;
```

Members

port
the port number.

ip_addr
the IP address.

46. TCPIP_NETIF_IPADDR_T

This structure lists the IP address information.

Syntax

```
typedef struct
{
    TCPIP_IPADDR_T    ipaddr;
    TCPIP_IPADDR_T    snmask;
    TCPIP_IPADDR_T    gateway;
    TCPIP_IPADDR_T    dns1;
    TCPIP_IPADDR_T    dns2;
} TCPIP_NETIF_IPADDR_T;
```

Members

ipaddr
The host IP. See TCPIP_IPADDR_T.
snmask
The subnet mask. See TCPIP_IPADDR_T.
gateway
The gateway. See TCPIP_IPADDR_T.
dns1
The primary DNS. See TCPIP_IPADDR_T.
dns2
The secondary DNS. See TCPIP_IPADDR_T.

47. TCPIP_IPADDR6_T

This structure shows the TCPIP IPv6 address(128 bit).

Syntax

```
typedef struct {
    union {
        uint8    u6_addr8[TCPIP_IP6_ADDR_LEN_BYTES];
        uint16   u6_addr16[TCPIP_IP6_ADDR_LEN_BYTES>>1];
```

```
            uint32    u6_addr32[TCPIP_IP6_ADDR_LEN_BYTES>>2];
        } u6_addr;
    } TCPIP_IPADDR6_T;
```

48. sci_sockaddr

In the Internet address family, this structure is used by Sockets to specify a local or remote endpoint address to which to connect a socket.

Syntax

```
struct sci_sockaddr
{
    unsigned short    family;
    unsigned short    port;
    unsigned long     ip_addr;
    char    sa_data[8];
};
```

Members

family
Address family.

port
Port number

ip_addr
Ip address

sa_data
Up to 14 bytes of direct address.

49. MERCURY_FD_SET_S

The definitions to support the select() function. These are about as UNIX-like as we can make 'em on embedded code. They are also fairly compatable with WinSock's select() definitions.

Syntax

```
typedef struct
{
    unsigned        fd_count;
    long            fd_array[12];
} MERCURY_FD_SET_S;
```

Members

fd_count

Number of sockets in the set.

fd_array

Array of sockets that are in the set.

50. FS_INIT_INFO_T

This structure defines how the file system uses flash.

Syntax

```
typedef struct
{
        uint32 startAddr;
        uint8    capacity;
        uint8    spiflashCap;
}FS_INIT_INFO_T;
```

Members

startAddr

This parameter represents the index of the file system sector, and the one block is 4K. Such as littlefs starts from at 2M, **startAddr** equals 512 (2M/4K = 512).

capacity

This represents the file system capacity.

spiflashCap

The total capacity of SPI flash.

Enumerations definition

1. DCAMERA_RETURN_VALUE_E

This enumeration shows the camera san possible error values.

Syntax

```
typedef enum
{
    DCAMERA_OP_SUCCESS = 0,
    DCAMERA_OP_ERROR,
    DCAMERA_OP_PARAM_ERR,
    DCAMERA_OP_NO_SENSOR_ERR,
    DCAMERA_OP_SENSOR_NOT_WORK_ERR,
    DCAMERA_OP_PREVIEW_ERR,
    DCAMERA_OP_IOCTL_ERR,
    DCAMERA_OP_SCAN_ERR,
    DCAMERA_OP_GET_SCAN_DATA_ERR,
    DCAMERA_OP_NO_ENOUGH_MEMORY,
    DCAMERA_OP_REVIEW_ERR,
    DCAMERA_OP_ISP_ERR,

    DCAMERA_OP_MAX = 0xFF,
}DCAMERA_RETURN_VALUE_E;
```

Members

Value	Description
DCAMERA_OP_SUCCESS	0, success
DCAMERA_OP_ERROR	1, normal error
DCAMERA_OP_PARAM_ERR	2, do scale set parameter error
DCAMERA_OP_NO_SENSOR_ERR	3, camera no sensor exist
DCAMERA_OP_SENSOR_NOT_WORK_ERR	4, init sensor error at open or set preview mode sensor error.
DCAMERA_OP_PREVIEW_ERR,	5, camera preview error.
DCAMERA_OP_IOCTL_ERR	6, the ioctl error
DCAMERA_OP_SCAN_ERR,	7, error while scanning
DCAMERA_OP_GET_SCAN_DATA_ERR	8, get the scan data error
DCAMERA_OP_NO_ENOUGH_MEMORY	9, there is not enough memory for camera scanning.
DCAMERA_OP_REVIEW_ERR	10, Camera review error
DCAMERA_OP_ISP_ERR	11, Failed to Open ISP Service

2. BARSCAN_MODE_VALUE_E

This enumeration shows the camera scan modes.

Syntax

```
typedef enum
{
    BARSCAN_MODE_CONTINUE = 0,
    BARSCAN_MODE_ONCE_SUSPEND = 0,
    BARSCAN_MODE_ONCE = 1,
    BARSCAN_MODE_REAL_CONTINUE = 2,
    BARSCAN_MODE_MAX,
}BARSCAN_MODE_VALUE_E;
```

Members

Value	Description
BARSCAN_MODE_CONTINUE	0, Scan to the code and enter suspend state, Rescan requires calling Cam_StartScan.
BARSCAN_MODE_ONCE_SUSPEND	0, The effect is the same as that of BARSCAN_MODE_CONTINUE.
BARSCAN_MODE_ONCE	1, In the mode, after successful code scanning, the code scanning module stops working and enters low-power mode.
BARSCAN_MODE_REAL_CONTINUE	2, Continues scan mode, if you frequently do other operations, such as screen brushing, you need to use with Cam_Suspend.

3. BARSCAN_FEATURE_CONFIGURE_E

This enumeration shows the bar scan feature mode.

Syntax

```
typedef enum
{
    BARSCAN_FEATURE_ALL = 0,
    BARSCAN_FEATURE_PAY,
    BARSCAN_FEATURE_BOX_PAY,
    BARSCAN_FEATURE_MAX,    //reserve
```

}BARSCAN_FEATURE_CONFIGURE_E;

Members

Value	Description
BARSCAN_FEATURE_ALL	0, The default value. Support all the modes
BARSCAN_FEATURE_PAY	1, Do not support pdf417, Datamatrix, location frame defiled code.
BARSCAN_FEATURE_BOX_PAY	2, Do not support pdf417, Datamatrix, location frame defiled code. And optimization experience of scanning at close range.

4. WAKE_LOCK_MODES

The enumeration shows the wake lock modes.

Syntax

```
typedef enum _wakelockmodes {
  LOCK_SCREEN,
  LOCK_SLEEP,
  LOCK_NONE
} WAKE_LOCK_MODES;
```

Members

LOCK_SCREEN

If a task hold the lock, the screen will keep on always and then the system will be unable to enter low-power state. In the mode, the automatic screen off time runs in cycles, until all such locks are released.

LOCK_SLEEP

If a task hold such lock, then the system will be unable to enter low-power state. So the system cannot come in deep sleep.

LOCK_NONE

No wake up any lock.

5. COM_PARITY_SET_E

Syntax

```
typedef enum
{
    PARITY_DISABLE = 0,
    PARITY_ENABLE
} COM_PARITY_SET_E;
```

6. DisplayOrientation

This enumerations lists the LCD possible display orientations.

Syntax

```
typedef enum{
    DMDO_0,
    DMDO_90,
    DMDO_180,
    DMDO_270
}DisplayOrientation;
```

7. MC_SPI_ID_E

This enumerations shows the range of the SPI id parameter values.

Syntax

```
typedef enum
{
    SPI_GROUP0_LOGIC_0 = 0,
    SPI_GROUP0_LOGIC_1 = 1,
    SPI_GROUP0_LOGIC_2 = 2,
    SPI_GROUP0_LOGIC_3 = 3,
    SPI_GROUP0_LOGIC_4 = 4,
    SPI_GROUP0_LOGIC_5 = 5
}MC_SPI_ID_E;
```

8. SPI_MODE_E

This enumerations list the spi mode.

Syntax

```
typedef enum
{
```

```
        CPOL0_CPHA0 = 0,        //sampling on rising edge, clk idle '0'
        CPOL0_CPHA1,            //sampling on falling edge, clk idle '0'
        CPOL1_CPHA0,            //sampling on falling edge, clk idle '1'
        CPOL1_CPHA1             //sampling on rising edge, clk idle '1'
    }SPI_MODE_E;
```

9. NLED_ID_E

This enumeration defines the nled id value.

Syntax

```
    typedef enum NLED_ID_E_TAG
    {
        LED_1,
        LED_2,
        LED_MAX_NUM
    }NLED_ID_E;
```

10. SYMBOL_TYPE_T

This enumeration defines the supported type of bar scanner. But if the type is PDF417 or DM, the result will return NONE.

Syntax

```
typedef enum {
    NONE        =    0,   /**< no symbol */
    PARTIAL     =    1,   /**< intermediate status */
    EAN8        =    8,   /**< EAN-8 */
    UPCE        =    9,   /**< UPC-E */
    ISBN10      =   10,   /**< ISBN-10 (from EAN-13)*/
    UPCA        =   12,   /**< UPC-A */
    EAN13       =   13,   /**< EAN-13 */
    ISBN13      =   14,   /**< ISBN-13 (from EAN-13).   */
    I25         =   25,   /**< Interleaved 2 of 5*/
    CODE39      =   39,   /**< Code 39.   */
    PDF417      =   57,   /**< PDF417.   */
    QRCODE      =   64,   /**< QR Code.   */
    CODE128     =  128,   /**< Code 128 */
    SYMBOL      = 0x00ff, /**< mask for base symbol type */
} symbol_type_t;
```

11. MERCURY_DIR_E

This enumeration lists the two states of the GPIO interrupt.

Syntax

```
typedef enum
{
    OUTPUT,
    INPUT,
    INVALID_DIR
} MERCURY_DIR_E;
```

Member

Value	Description
OUTPUT	GPIO direction is output
INPUT	GPIO direction is input

12. MERCURY_INTERTUPT_MODE_E

This enumeration list the interrupt mode possible values.

Syntax

```
typedef enum
{
    LEVEL,
    RISING_EDGE,
    FALLING_EDGE,
    BOTH_EDGE,
    NO_INT,
    INVALID_INT
} MERCURY_INTERTUPT_MODE_E;
```

Member

Value	Description
LEVEL	Level trigger interrupt
RISING_EDGE	Rising edge trigger interrupt
FALLING_EDGE	Failing edge trigger interrupt
BOTH_EDGE	Both rising edge and falling edge trigger interrupt
NO_INT	No trigger interrupt

13. NVITEM_ERROR_E

This enumeration list errors related to NV.

Syntax

```
typedef enum _NVITEM_ERROR {
    NVERR_NONE    = 0,              /* Success */
    NVERR_SYSTEM,                   /* System error, e.g. hardware failure */
    NVERR_INVALID_PARAM,
    NVERR_NO_ENOUGH_RESOURCE,
    NVERR_NOT_EXIST,
    NVERR_ACCESS_DENY,
    NVERR_INCOMPATIBLE,
    NVERR_NOT_OPENED
}NVITEM_ERROR_E;
```

Members

Value	Description
NVERR_NONE = 0	The item is written successfully.
NVERR_SYSTEM	System error, e.g. hardware failure
NVERR_INVALID_PARAM	Parameters are invalid, e.g. buf_ptr is NULL or Identifier is invalid.
NVERR_NO_ENOUGH_RESOURCE	There is no enough resource to complete this operation, e.g. no enough space on the medium.
NVERR_NOT_EXIST	The nv item does not exist.
NVERR_ACCESS_DENY	The nv item access denied
NVERR_INCOMPATIBLE	The nv item is incompatible.
NVERR_NOT_OPENED	The nv item can't be opened

14. KEYPAD_UID_E

This enumeration defines number of keypad ids supported.

Syntax

```
typedef enum    _KEYPAD_UID_E
{
    UID_1 = 0,
    UID_2,
    UID_3,
    UID_4,
```

```
        UID_5,
        UID_6,
        UID_7,
        UID_8,
        UID_9,
        UID_10,
        UID_MAX,
}KEYPAD_UID_E;
```

15. MERCURY_PHONE_PLMN_STATUS_E

This enumerations definition plmn status.

Syntax

```
typedef enum
{
        PLMN_NO_SERVICE = 0 ,                   // no service
        PLMN_NORMAL_GSM_ONLY = 0x01 ,           // voice service available
        PLMN_NORMAL_GPRS_ONLY = 0x02 ,          // data service available
        PLMN_NORMAL_GSM_GPRS_BOTH = 0x03 ,// voice and data service
available

        PLMN_NORMAL_CS_ONLY = 0x01 ,            // voice service available
        PLMN_NORMAL_PS_ONLY = 0x02 ,            // data service available
        PLMN_NORMAL_CS_PS_BOTH = 0x03 ,// voice and data service available

        PLMN_EMERGENCY_ONLY = 0x04,             // emergency service available

        PLMN_EMERGENCY_SIM_INVALID = 0x05,      /* emergency; MM in
limited service state and
                                                no further PLMN
access allowed until power
                                                off or new SIM inserted
*/
        PLMN_EMERGENCY_GPRS_ONLY = 0x06,        // data service available
but emergency; MM in limited service state
        PLMN_EMERGENCY_SIM_INVALID_GPRS_ONLY = 0x07,  /* data service
availabe but emergency; MM in limited service state and
                                                no further PLMN
access allowed until power
                                                off or new SIM
inserted */
```

```
    PLMN_REGISTER_SERVICE = 0x08,        //attaching after camping on
    PLMN_REGISTER_GPRS_ONLY = 0x09, // data service available but attaching;
    PLMN_FULL_PS_SERVICE = 0x0A          /*full PS service, no cs service*/

} MERCURY_PHONE_PLMN_STATUS_E;
```

16. MERCURY_ATTACH_STATE_E

Syntax

```
typedef enum
{
    MN_INVALID_STATE, /* this field is invalid */
    MN_ATTACHED_STATE, /* cs or ps has been attached */
    MN_DETACHED_SATE, /* cs or ps has been detached */
    MN_NO_SERVICE    /* no service for cs or ps */
} MERCURY_ATTACH_STATE_E;
```

17. FILESYS_CAPACITY_E

This enumerations lists the size of the file system that can be set.

Syntax

```
typedef enum
{
    FILESYS_CAPACITY_1M = 1,
    FILESYS_CAPACITY_2M,
    FILESYS_CAPACITY_3M,
    FILESYS_CAPACITY_4M,
    FILESYS_CAPACITY_5M,
    FILESYS_CAPACITY_6M,
    FILESYS_CAPACITY_7M,
    FILESYS_CAPACITY_8M,
    FILESYS_CAPACITY_9M,
    FILESYS_CAPACITY_10M,
    FILESYS_CAPACITY_11M,
    FILESYS_CAPACITY_12M,
```

```
    FILESYS_CAPACITY_13M,
    FILESYS_CAPACITY_14M,
    FILESYS_CAPACITY_15M,
    FILESYS_CAPACITY_16M,
    FILESYS_CAPACITY_MAX
}FILESYS_CAPACITY_E;
```

Members

Value	Description
FILESYS_CAPACITY_1M	Set the capacity of 1M
FILESYS_CAPACITY_2M	Set the capacity of 2M
FILESYS_CAPACITY_3M	Set the capacity of 3M
FILESYS_CAPACITY_4M	Set the capacity of 4M
FILESYS_CAPACITY_5M	Set the capacity of 5M
FILESYS_CAPACITY_6M	Set the capacity of 6M
FILESYS_CAPACITY_7M	Set the capacity of 7M
FILESYS_CAPACITY_8M	Set the capacity of 8M
FILESYS_CAPACITY_9M	Set the capacity of 9M
FILESYS_CAPACITY_10M	Set the capacity of 10M
FILESYS_CAPACITY_11M	Set the capacity of 11M
FILESYS_CAPACITY_12M	Set the capacity of 12M
FILESYS_CAPACITY_13M	Set the capacity of 13M
FILESYS_CAPACITY_14M	Set the capacity of 14M
FILESYS_CAPACITY_15M	Set the capacity of 15M
FILESYS_CAPACITY_16M	Set the capacity of 16M

18. NOTIFY_CLASS_E

The enumeration shows the receive message classes by NotifyCallback.

Syntax

```
typedef enum
{
    NOTIFY_CLASS_PDP = 0,
    NOTIFY_CLASS_SMS,
    NOTIFY_CLASS_TEL,
    NOTIFY_CLASS_SOCKET,
    NOTIFY_CLASS_DNS,
    NOTIFY_CLASS_CHARGE,
    NOTIFY_CLASS_SCREEN,
    NOTIFY_CLASS_POWER,
    NOTIFY_CLASS_BARSCAN,
```

```
        NOTIFY_CLASS_TTS,
        NOTIFY_CLASS_SYSTEM,
        NOTIFY_CLASS_AUDIO,
        NOTIFY_CLASS_STK,
        NOTIFY_CLASS_MAX
}NOTIFY_CLASS_E;
```

Members

Value	Description
NOTIFY_CLASS_PDP	This class tells the processing state of the PDP. And corresponding notify id is PDP_NOTIFY_ID_E.
NOTIFY_CLASS_SMS	This class notify the sms processing result. And corresponding notify id is SMS_NOTIFY_ID_E.
NOTIFY_CLASS_TEL	This class informs the status of the phone.
NOTIFY_CLASS_SOCKET	This class informs the status of the socket.
NOTIFY_CLASS_DNS	This class informs the analytical results of DNS.
NOTIFY_CLASS_CHARGE	This class informs the charge status of the battery.
NOTIFY_CLASS_SCREEN	In Screen Unlock mode, short press power key, this class notify the screen on/off status.
NOTIFY_CLASS_POWER	This class notify power key status, such as long press. And in Screen Lock mode, short press power key, that will notify the short press notify. See MC_POWER_NOTIFY_ID_E.
NOTIFY_CLASS_BARSCAN	This class informs the bar scan status. And corresponding notify id is MC_BARSCAN_NOTIFY_ID_E.
NOTIFY_CLASS_TTS	This class informs the TTS play result.
NOTIFY_CLASS_SYSTEM	This class informs the system status, it contains SIM status, PS and so on.
NOTIFY_CLASS_AUDIO	This class informs the audio play status. it contains dtmf play result. See MC_AUDIO_NOTIFY_ID_E.
NOTIFY_CLASS_STK	This class informs the get esim id status.

19. PDP_NOTIFY_ID_E

The enumeration notify the pdp operate type.

Syntax

```
typedef enum
{
        NOTIFY_ID_ACT_SUCESS = 0,
        NOTIFY_ID_ACT_FAILE,
        NOTIFY_ID_DEACT_SUCESS,
```

```
        NOTIFY_ID_DEACT_FAILE,
        NOTIFY_ID_DEACT_BY_NET,
        NOTIFY_ID_ATTACH_SUCCESS,
        NOTIFY_ID_ATTACH_FAILE,
        NOTIFY_ID_DEATTACH_SUCCESS,
        NOTIFY_ID_DEATTACH_FAILE,
        NOTIFY_ID_DEATTACH_BY_NET,
        NOTIFY_ID_DEACT,
}PDP_NOTIFY_ID_E;
```

Members

Value	Description
NOTIFY_ID_ACT_SUCESS	Successful activation of PDP.
NOTIFY_ID_ACT_FAILE	Failed to active PDP.
NOTIFY_ID_DEACT_SUCESS	PDP deactivate successful.
NOTIFY_ID_DEACT_FAILE	Failed to deactivate PDP.
NOTIFY_ID_DEACT_BY_NET	The PDP is deactivation by network.
NOTIFY_ID_ATTACH_SUCCESS	Gprs attach success.
NOTIFY_ID_ATTACH_FAILE	Gprs attach fail.
NOTIFY_ID_DEATTACH_SUCCESS	Gprs detach success.
NOTIFY_ID_DEATTACH_FAILE	Gprs detach fail.
NOTIFY_ID_DEATTACH_BY_NET	Gprs detach by network.
NOTIFY_ID_DEACT	Reserved, user can't use it.

20. SMS_NOTIFY_ID_E

The enumeration notify the sms operate result type.

Syntax

```
typedef enum
{
        NOTIFY_ID_SEND_SUCESS= 0,
        NOTIFY_ID_SEND_FAILE,
        NOTIFY_ID_READ_TEXT,
        NOTIFY_ID_READ_PDU,
        NOTIFY_ID_RCV_SMS,
        NOTIFY_ID_SMS
}SMS_NOTIFY_ID_E;
```

Members

Value	Description
NOTIFY_ID_SEND_SUCESS	It notifies app that the SMS success.

NOTIFY_ID_SEND_FAILE	It notifies app that the SMS failure.
NOTIFY_ID_READ_TEXT	It notifies app that The SMS reading result and include the read text data. SMS data structure see SMS_REC_TEXT_S.
NOTIFY_ID_READ_PDU	It notifies app that The SMS reading result and include the read data. SMS data structure see SMS_REC_PDU_S.
NOTIFY_ID_RCV_SMS	It notifies app that received SMS, and upload received SMS position id.
NOTIFY_ID_SMS	Reserved, user can't use it.

21. SOCKET_NOTIFY_ID_E

The enumeration notify the socket operate result type.

Syntax

```
typedef enum
{
        NOTIFY_ID_SOCKET_CONNECT = 0,
        NOTIFY_ID_SCOKET_READ,
        NOTIFY_ID_SCOKET_WRITE,
        NOTIFY_ID_SCOKET_CLOSE,
        NOTIFY_ID_SOCKET_FULLCLOSE,      //local close completed reporting
        NOTIFY_ID_SCOKET
}SOCKET_NOTIFY_ID_E;
```

Members

Value	Description
NOTIFY_ID_SOCKET_CONNECT	Socket connect success, and upload socket id and error code. The first 4 bytes are socket id, the last 4 bytes are error code.
NOTIFY_ID_SCOKET_READ	It notifies user that the device received the data sent by the server, and now you can use **SocketRecv** to read the data. And upload socket id and error code. The first 4 bytes are socket id, the last 4 bytes are error code.
NOTIFY_ID_SCOKET_WRITE	Ignored
NOTIFY_ID_SCOKET_CLOSE	Remote close socket completed reporting. And upload socket id and error code. The first 4 bytes are socket id, the last 4 bytes are error code.
NOTIFY_ID_SOCKET_FULLCLOSE	local close completed reporting. And upload socket id and error code. The first 4 bytes are socket id, the

	last 4 bytes are error code.
NOTIFY_ID_SCOKET	Reserved, user can't use it.

22. MC_SCREEN_NOTIFY_ID_E

The enumeration notify the screen statue. This message notification type will only occur when the screen is unlocked.

Syntax

```
typedef enum
{
    NOTIFY_ID_SCREEN_ON= 0,
    NOTIFY_ID_SCREEN_OFF,
    NOTIFY_ID_SCREEN
}MC_SCREEN_NOTIFY_ID_E;
```

Members

Value	Description
NOTIFY_ID_SCREEN_ON	Notify the screen on, And force the screen to be on at the bottom layer.
NOTIFY_ID_SCREEN_OFF	Notify the screen on, And force the screen to be off at the bottom layer.

23. MC_POWER_NOTIFY_ID_E

The enumeration notify the power key statue.

Syntax

```
typedef enum
{
    NOTIFY_ID_POWERKEY_LONGPRESS= 0,
    NOTIFY_ID_POWERKEY_SHORTPRESS,
    NOTIFY_ID_POWER_MSG_MAX_NUM
}MC_POWER_NOTIFY_ID_E;
```

Members

Value	Description
NOTIFY_ID_POWERKEY_LONGPRESS	This message notification type will occur when a long press on the power key.
NOTIFY_ID_POWERKEY_SHORTPRESS	This message notification type will occur

	when a short press on the power key and screen must be lock.

24. MC_BARSCAN_NOTIFY_ID_E

The enumeration lists the bar scan possible results.

Syntax

```
typedef enum
{
    NOTIFY_ID_BARSCAN_INITED = 0,
    NOTIFY_ID_BARSCAN_PRESCAN_DONE,
    NOTIFY_ID_BARSCAN_SCANING,
    NOTIFY_ID_BARSCAN_SCANED_SUCCESS,
    NOTIFY_ID_BARSCAN_SCANED_FAILED,
    NOTIFY_ID_BARSCAN_ID_MAX
}MC_BARSCAN_NOTIFY_ID_E;
```

Members

Value	Description
NOTIFY_ID_BARSCAN_INITED	Bar scan already initialized.
NOTIFY_ID_BARSCAN_PRESCAN_DONE	Pre scanning has been completed.
NOTIFY_ID_BARSCAN_SCANING	Camera is scanning.
NOTIFY_ID_BARSCAN_SCANED_SUCCESS	Bar scan successful.
NOTIFY_ID_BARSCAN_SCANED_FAILED	Bar scan failure.

25. MC_TTS_NOTIFY_ID_E

The enumeration lists the TTS playing possible result.

Syntax

```
typedef enum
{
    NOTIFY_ID_TTS_PLAY_COMPLETE= 0,
    NOTIFY_ID_TTS_MSG_MAX
}MC_TTS_NOTIFY_ID_E;
```

Members

Value	Description
NOTIFY_ID_TTS_PLAY_COMPLETE	It notifies that TTS plays complete.

26. SYSTEM_NOTIFY_ID_E

The enumeration lists the system possible status.

Syntax

```
typedef enum
{
    NOTIFY_ID_SIM_READY= 0,
    NOTIFY_ID_SIM_NOT_READY,
    NOTIFY_ID_PS_POWER_ON,
    NOTIFY_ID_PS_POWER_OFF,
    NOTIFY_ID_SYSTEM_MSG_MAX
} SYSTEM_NOTIFY_ID_E;
```

Members

Value	Description
NOTIFY_ID_SIM_READY	Notify SIM card is ready.
NOTIFY_ID_SIM_NOT_READY	SIM card is not ready.
NOTIFY_ID_PS_POWER_ON	Notify that ps power on.
NOTIFY_ID_PS_POWER_OFF	Notify that ps power off.
NOTIFY_ID_SYSTEM_MSG_MAX	Reserved

27. TEL_NOTIFY_ID_E

The enumeration notify the results of the telephony operation.

Syntax

```
typedef enum
{
    NOTIFY_ID_CALL_START=0,
    NOTIFY_ID_INCOMING_RING,
    NOTIFY_ID_REMOTE_HANG,
    NOTIFY_ID_REMOTE_BUSY,
    NOTIFY_ID_REMOTE_NO_ANSWER,
    NOTIFY_ID_TEL
}TEL_NOTIFY_ID_E;
```

Members

Value	Description

NOTIFY_ID_CALL_START	Telephony start to connect.
NOTIFY_ID_INCOMING_RING	Telephony incoming ring.
NOTIFY_ID_REMOTE_HANG	Telephony remote hang.
NOTIFY_ID_REMOTE_BUSY	Telephony remote busy.
NOTIFY_ID_REMOTE_NO_ANSWER	Telephony remote no answer.
NOTIFY_ID_TEL	Reserved.

28. DNS_NOTIFY_ID_E

The enumeration notify the resolve DNS result type.

Syntax

```
typedef enum
{
    NOTIFY_ID_DNS_SUCESS= 0,
    NOTIFY_ID_DNS_FAILE,
    NOTIFY_ID_DNS
}DNS_NOTIFY_ID_E;
```

Members

Value	Description
NOTIFY_ID_DNS_SUCESS	Resolve DNS successful. 8 bytes of the data returned by the message, The first 4 bytes are request id, the last 4 bytes are IP address.
NOTIFY_ID_DNS_FAILE	Resolve DNS failure. Message notification returned to request id.
NOTIFY_ID_DNS	Reserved.

29. MC_AUDIO_NOTIFY_ID_E

The enumeration notifies DTMF play result.

Syntax

```
typedef enum
{
    NOTIFY_ID_DTMF_PLAY_COMPLETE= 0,
    NOTIFY_ID_STONE_PLAY_COMPLETE,
    NOTIFY_ID_PCM_PLAY_COMPLETE,
    NOTIFY_ID_AUDIO_MSG_MAX,
}MC_AUDIO_NOTIFY_ID_E;
```

Members

Value	Description
NOTIFY_ID_DTMF_PLAY_COMPLETE	Notify dtmf play complete.
NOTIFY_ID_STONE_PLAY_COMPLETE	Notify single tone play complete.
NOTIFY_ID_PCM_PLAY_COMPLETE	Notify pcm file play complete.

30. PDP_ID_E

The enumeration lists the identity of the PDP.

Syntax

```
typedef enum
{
    PDP_ID0=1,
    PDP_ID1,
    PDP_ID2
}PDP_ID_E;
```

members

Value	Description
PDP_ID0	The default value, and only support this.
PDP_ID1	Reserved.
PDP_ID2	Reserved.

31. SIM_INFO_E

The enumeration list the available SIM card information.

Syntax

```
typedef enum
{
    SIM_STATE_E = 0,
    SIM_IMSI_E,
    SIM_CCID_E,
    SIM_ALL,
    SIM_MAX
}SIM_INFO_E;
```

members

Value	Description
SIM_STATE_E	Just get the sim card status
SIM_IMSI_E	get the IMSI number and sim card status
SIM_CCID_E	get the CCID number and sim card status
SIM_ALL	get the IMSI , CCID number and sim card status.

32. SMS_CHARACTER_SET_TYPE_E

This enumeration set the sms character type.

Syntax

```
typedef enum
{
    ATC_CHSET_IRA = 0,
    ATC_CHSET_GSM,
    ATC_CHSET_HEX,
    ATC_CHSET_UCS2,
    ATC_CHSET_MAX_NUM
} SMS_CHARACTER_SET_TYPE_E;
```

members

Value	Description
ATC_CHSET_IRA	International reference alphabet
ATC_CHSET_GSM	GSM 7 bit default alphabet
ATC_CHSET_HEX	Character strings consist only of hexadecimal numbers from 00 to FF
ATC_CHSET_UCS2	16-bit universal multiple-octet coded character set

33. SOCKET_TYPE_E

This enumeration explain the type of the socket.

Syntax

```
typedef enum
{
    SOCKET_TYPE_TCP = 0,
```

```
        SOCKET_TYPE_UDP,
        SOCKET_TYPE_NULL
}SOCKET_TYPE_E;
```

34. AUDIO_DEVICE_MODE_TYPE_E

This Enumerations definition brief Audio device mode type list.

Syntax

```
typedef enum
{
    AUDIO_DEVICE_MODE_HANDHOLD,
    AUDIO_DEVICE_MODE_HANDFREE,
    AUDIO_DEVICE_MODE_EARPHONE,
    AUDIO_DEVICE_MODE_EARFREE,
    AUDIO_DEVICE_MODE_TVOUT,
    AUDIO_DEVICE_MODE_BLUEPHONE,
    AUDIO_DEVICE_MODE_MAX
}AUDIO_DEVICE_MODE_TYPE_E;
```

35. MC_CHR_NOTIFY_ID_E

The enumeration shows the charge status of the battery.

Syntax

```
typedef enum
{
        NOTIFY_ID_CHR_CAP_IND = 0x1,
        NOTIFY_ID_CHR_CHARGE_START_IND,
        NOTIFY_ID_CHR_CHARGE_END_IND,
        NOTIFY_ID_CHR_WARNING_IND,
        NOTIFY_ID_CHR_SHUTDOWN_IND,
        NOTIFY_ID_CHR_CHARGE_FINISH,
        NOTIFY_ID_CHR_CHARGE_DISCONNECT,
        NOTIFY_ID_CHR_CHARGE_FAULT,
        NOTIFY_ID_CHR_CHARGE_PLUG_IN_MSG,
        NOTIFY_ID_CHR_CHARGE_PLUG_OUT_MSG,
        NOTIFY_ID_CHR_MSG_MAX_NUM
} MC_CHR_NOTIFY_ID_E;
```

Members

Value	Description
NOTIFY_ID_CHR_CAP_IND	Reserved .(Notify the battery's capacity)
NOTIFY_ID_CHR_CHARGE_START_IND	start the charge process.
NOTIFY_ID_CHR_CHARGE_END_IND	Reserved.
NOTIFY_ID_CHR_WARNING_IND	the capacity is low, should charge.
NOTIFY_ID_CHR_SHUTDOWN_IND	the capacity is very low and must shutdown.
NOTIFY_ID_CHR_CHARGE_FINISH	the charge has been completed.
NOTIFY_ID_CHR_CHARGE_DISCONNECT	the charge be disconnect
NOTIFY_ID_CHR_CHARGE_FAULT	the charge fault, maybe the voltage of charge is too low.
NOTIFY_ID_CHR_CHARGE_PLUG_IN_MSG	USB plug in
NOTIFY_ID_CHR_CHARGE_PLUG_OUT_MSG	USB plug out

36. MERCURY_AUDIO_MODE_TYPE_E

The enumeration shows the audio mode types.

Syntax

```
typedef enum
{
  AUDIO_MODE_HANDHOLD,
  AUDIO_MODE_HANDFREE,
  AUDIO_MODE_EARPHONE,
  AUDIO_MODE_EARFREE,
  AUDIO_MODE_MAX
}MERCURY_AUDIO_MODE_TYPE_E;
```

Members

Value	Description
AUDIO_MODE_HANDHOLD	0, handhold mode
AUDIO_MODE_HANDFREE	1, Hands-Free mode
AUDIO_MODE_EARPHONE	2, earphone mode
AUDIO_MODE_EARFREE	3, hands-free and headset mode

37. AUDIO_VOLUME_LEVEL_E

The enumeration shows the volume levels.

Syntax

```
typedef enum
{
    AUDIO_VOLUME_LEVEL1 = 1,
    AUDIO_VOLUME_LEVEL2 = 2,
    AUDIO_VOLUME_LEVEL3 = 3,
    AUDIO_VOLUME_LEVEL4 = 4,
    AUDIO_VOLUME_LEVEL5 = 5,
    AUDIO_VOLUME_LEVEL6 = 6,
    AUDIO_VOLUME_LEVEL7 = 7,
    AUDIO_VOLUME_LEVEL8 = 8,
    AUDIO_VOLUME_LEVEL9 = 9
}AUDIO_VOLUME_LEVEL_E;
```

38. MERCURY_DTMF_TONE_ID_E

The enumeration shows the dtmf tone id.

Syntax

```
typedef enum
{
    DTMF_One,          // 1
    DTMF_Two,          // 2
    DTMF_Three,        // 3
    DTMF_letterA,      // A
    DTMF_Four,         // 4
    DTMF_Five,         // 5
    DTMF_Six,          // 6
    DTMF_letterB,      // B
    DTMF_Seven,        // 7
    DTMF_Eight,        // 8
    DTMF_Nine,         // 9
    DTMF_letterC,      // C
    DTMF_Star,         // *
    DTMF_Zero,         // 0
    DTMF_Pond,         // #
    DTMF_letterD,      // D
    DTMF_MAX_ID        // Reserved, user can't use it.
} MERCURY_DTMF_TONE_ID_E;
```

39. ADC_ID_E

The enumeration shows the ADC id.

Syntax

```
typedef enum
{
    ADC_ID_0 = 0, // Get battery temperature ADC value
    ADC_ID_1 = 1,
    ADC_ID_MAX
} ADC_ID_E;
```

40. ADC_SCALE_E

The enumeration shows the ADC supports scale.

Syntax

```
typedef enum
{
    ADC_SCALE_12BIT_3V = 0,
    ADC_SCALE_12BIT_1V2,
    ADC_SCALE_MAX
}ADC_SCALE_E;
```

Members

Value	Description
ADC_SCALE_12BIT_3V	0, ADC is12bit and the reference voltage is 3
ADC_SCALE_12BIT_1V2	1, ADC is12bit and the reference voltage is 1.2V

41. PCO_AUTH_TYPE_E

The enumeration shows the network authentication mode.

Syntax

```
typedef enum
{
    AUTH_PAP        = 0,
    AUTH_CHAP       = 1,
    AUTH_PAP_CHAP   = 2,
    AUTH_NONE = 3
```

} PCO_AUTH_TYPE_E;

Members

Value	Description
AUTH_PAP	PAP authentication
AUTH_CHAP	CHAP authentication
AUTH_PAP_CHAP	PAP and CHAP authentication
AUTH_NONE	No authentication

42. QRecLevel

This enumeration shows the level of error correction for QR code.

Syntax

```
typedef enum {
    QR_ECLEVEL_L = 0, ///< lowest
    QR_ECLEVEL_M,
    QR_ECLEVEL_Q,
    QR_ECLEVEL_H,        ///< highest
    QR_ECLEVEL_MAX = 0x7fffffff
} QRecLevel;
```

43. LITTLEFS_CAPACITY_E

This enumerations lists the size of the littlefs that can be set.

Syntax

```
typedef enum
{
    LITTLEFS_CAPACITY_1M = 1,
    LITTLEFS_CAPACITY_2M,
    LITTLEFS_CAPACITY_3M,
    LITTLEFS_CAPACITY_4M,
    LITTLEFS_CAPACITY_5M,
    LITTLEFS_CAPACITY_6M,
    LITTLEFS_CAPACITY_7M,
    LITTLEFS_CAPACITY_8M,
    LITTLEFS_CAPACITY_9M,
    LITTLEFS_CAPACITY_10M,
    LITTLEFS_CAPACITY_11M,
```

```
        LITTLEFS_CAPACITY_12M,
        LITTLEFS_CAPACITY_13M,
        LITTLEFS_CAPACITY_14M,
        LITTLEFS_CAPACITY_15M,
        LITTLEFS_CAPACITY_16M,
        LITTLEFS_CAPACITY_MAX,
    } LITTLEFS_CAPACITY_E;
```

Members

Value	Description
LITTLEFS_CAPACITY_1M	Set the capacity of 1M
LITTLEFS _CAPACITY_2M	Set the capacity of 2M
LITTLEFS _CAPACITY_3M	Set the capacity of 3M
LITTLEFS _CAPACITY_4M	Set the capacity of 4M
LITTLEFS _CAPACITY_5M	Set the capacity of 5M
LITTLEFS _CAPACITY_6M	Set the capacity of 6M
LITTLEFS _CAPACITY_7M	Set the capacity of 7M
LITTLEFS _CAPACITY_8M	Set the capacity of 8M
LITTLEFS _CAPACITY_9M	Set the capacity of 9M
LITTLEFS _CAPACITY_10M	Set the capacity of 10M
LITTLEFS _CAPACITY_11M	Set the capacity of 11M
LITTLEFS _CAPACITY_12M	Set the capacity of 12M
LITTLEFS _CAPACITY_13M	Set the capacity of 13M
LITTLEFS _CAPACITY_14M	Set the capacity of 14M
LITTLEFS _CAPACITY_15M	Set the capacity of 15M
LITTLEFS _CAPACITY_16M	Set the capacity of 16M

44. MC_STK_NOTIFY_ID_E

This enumerations lists information about STK.

Syntax

```
typedef enum
{
    NOTIFY_ID_STK_REFRESH_IND= 0,
    NOTIFY_ID_STK_EID_GET_SUCCESS,
    NOTIFY_ID_STK_EID_GET_FAILE,
    NOTIFY_ID_STK_MSG_MAX = 0x7fffffff
}MC_STK_NOTIFY_ID_E;
```

Members

Value	Description

NOTIFY_ID_STK_EID_GET_SUCCESS	Get ESIM ID success.
NOTIFY_ID_STK_EID_GET_FAILE	Get ESIM ID fail.

45. AMOI_SYMBOL_TYPE_T

This enumeration lists the symbol of bar code.

Syntax

```
typedef enum {
    AMOI_SYMBOL_GOODS = 0,
    AMOI_SYMBOL_I25,
    AMOI_SYMBOL_DATABAR,
    AMOI_SYMBOL_CODABAR,
    AMOI_SYMBOL_CODE39 ,
    AMOI_SYMBOL_PDF417 ,
    AMOI_SYMBOL_QRCODE ,
    AMOI_SYMBOL_CODE93 ,
    AMOI_SYMBOL_CODE128 ,
    AMOI_SYMBOL_MAX ,
    MERCURY_ENUM_AMOI_SYMBOL_MAX = 0x7fffffff
} AMOI_SYMBOL_TYPE_T;
```

Members

Value	Description
AMOI_SYMBOL_GOODS	EAN ISBN UPCA etc
AMOI_SYMBOL_I25	Interleaved 2 of 5
AMOI_SYMBOL_DATABAR	GS1 DataBar (RSS).
AMOI_SYMBOL_CODABAR	Codabar
AMOI_SYMBOL_CODE39	Code 39.
AMOI_SYMBOL_PDF417	PDF417. no support
AMOI_SYMBOL_QRCODE	QR Code.
AMOI_SYMBOL_CODE93	Code 93
AMOI_SYMBOL_CODE128	Code 128

RGB565 Color Index Table

The following table shows the rgb565 palette, there are 242 colors in all.

Color	Palette Index	RGB565 Value
	MERCURY_BLACK	0x0000
	MERCURY_DIMGRAY	0x6B4D
	MERCURY_GRAY	0x8410
	MERCURY_DARK_GRAY	0xAD55
	MERCURY_SILVER	0xC618
	MERCURY_LIGHT_GRAY	0xD69A
	MERCURY_GAINSBORO	0xDEFB
	MERCURY_WHITE_SMOKE	0xF7BE
	MERCURY_WHITE	0xFFFF
	MERCURY_SNOW	0xFFDF
	MERCURY_IRON_GRAY	0x62CA
	MERCURY_SAND_BEIGE	0xE618
	MERCURY_ROSY_BROWN	0xBC71
	MERCURY_LIGHT_CORAL	0xF410
	MERCURY_INDIAN_RED	0xCAEB
	MERCURY_BROWN	0xA145
	MERCURY_FIRE_BRICK	0xB104
	MERCURY_MAROON	0x8000
	MERCURY_DARK_RED	0x8800
	MERCURY_STRONG_RED	0xE000
	MERCURY_RED	0xF800
	MERCURY_PERSIMMON	0xFA68
	MERCURY_MISTY_ROSE	0xFF3C
	MERCURY_SALMON	0xFC0E
	MERCURY_SCARLET	0xF920

	MERCURY_TOMATO	0xFB08
	MERCURY_DARK_SALMON	0xECAF
	MERCURY_CORAL	0xFBEA
	MERCURY_ORANGE_RED	0xFA20
	MERCURY_LIGHT_SALMON	0xFD0F
	MERCURY_VERMILION	0xFA60
	MERCURY_SIENNA	0xA285
	MERCURY_TROPICAL_ORANGE	0xFC06
	MERCURY_CAMEL	0xA348
	MERCURY_APRICOT	0xE4CC
	MERCURY_COCONUT_BROWN	0x48E0
	MERCURY_SEASHELL	0xFFBD
	MERCURY_SADDLE_BROWN	0x8A22
	MERCURY_CHOCOLATE	0xD343
	MERCURY_BURNT_ORANGE	0xCAA0
	MERCURY_SUN_ORANGE	0xFB80
	MERCURY_PEACH_PUFF	0xFED7
	MERCURY_SAND_BROWN	0xF52C
	MERCURY_BRONZE	0xBB86
	MERCURY_LINEN	0xFF9C
	MERCURY_HONEY_ORANGE	0xFD8C
	MERCURY_PERU	0xCC27
	MERCURY_SEPIA	0x7202
	MERCURY_OCHER	0xCBA4
	MERCURY_BISQUE	0xFF38
	MERCURY_TANGERINE	0xF420
	MERCURY_DARK_ORANGE	0xFC60
	MERCURY_ANTIQUE_WHITE	0xFF5A
	MERCURY_TAN	0xD5B1

	MERCURY_BURLY_WOOD	0xDDD0
	MERCURY_BLANCHED_ALMOND	0xFF59
	MERCURY_NAVAJO_WHITE	0xFEF5
	MERCURY_MARIGOLD	0xFCC0
	MERCURY_PAPAYA_WHIP	0xFF7A
	MERCURY_PALE_OCRE	0xCD91
	MERCURY_KHAKI	0x9B43
	MERCURY_MOCCASIN	0xFF36
	MERCURY_OLD_LACE	0xFFBC
	MERCURY_WHEAT	0xF6F6
	MERCURY_PEACH	0xFF36
	MERCURY_ORANGE	0xFD20
	MERCURY_FLORAL_WHITE	0xFFDE
	MERCURY_GOLDENROD	0xDD24
	MERCURY_DARK_GOLDENROD	0xBC21
	MERCURY_COFFEE	0x49C0
	MERCURY_JASMINE	0xE60B
	MERCURY_AMBER	0xFDE0
	MERCURY_CORNSILK	0xFFDB
	MERCURY_CHROME_YELLOW	0xE5C0
	MERCURY_GOLDEN	0xFEA0
	MERCURY_LEMON_CHIFFON	0xFFD9
	MERCURY_LIGHT_KHAKI	0xF731
	MERCURY_PALE_GOLDENROD	0xEF55
	MERCURY_DARK_KHAKI	0xBDAD
	MERCURY_MIMOSA	0xE6C6
	MERCURY_CREAM	0xFFFA
	MERCURY_IVORY	0xFFFE
	MERCURY_BEIGE	0xF7BB

	MERCURY_LIGHT_YELLOW	0xFFFC
	MERCURY_LIGHT_GOLDENROD_YELLOW	0xFFDA
	MERCURY_CHAMPAGNE_YELLOW	0xFFF3
	MERCURY_MUSTARD	0xCE69
	MERCURY_MOON_YELLOW	0xFFE9
	MERCURY_OLIVE	0x8400
	MERCURY_CANARY_YELLOW	0xFFE0
	MERCURY_YELLOW	0xFFE0
	MERCURY_MOSS_GREEN	0x6BA4
	MERCURY_LIGHT_LIME	0xCFE0
	MERCURY_OLIVE_DRAB	0x6C64
	MERCURY_YELLOW_GREEN	0x9E66
	MERCURY_DARK_OLIVE_GREEN	0x5345
	MERCURY_APPLE_GREEN	0x8F20
	MERCURY_GREEN_YELLOW	0xAFE5
	MERCURY_GRASS_GREEN	0x9F29
	MERCURY_LAWN_GREEN	0x7FE0
	MERCURY_CHARTREUSE	0x7FE0
	MERCURY_FOLIAGE_GREEN	0x75C7
	MERCURY_FRESH_LEAVES	0x9FE9
	MERCURY_BRIGHT_GREEN	0x67E0
	MERCURY_COBALT_GREEN	0x67EB
	MERCURY_HONEYDEW	0xF7FE
	MERCURY_DARK_SEA_GREEN	0x8DF1
	MERCURY_LIGHT_GREEN	0x9772
	MERCURY_PALE_GREEN	0x9FD3
	MERCURY_IVY_GREEN	0x35E6
	MERCURY_FOREST_GREEN	0x2444
	MERCURY_LIME_GREEN	0x3666

	MERCURY_DARK_GREEN	0x0320
	MERCURY_GREEN	0x0400
	MERCURY_LIME	0x07E0
	MERCURY_MALACHITE	0x2605
	MERCURY_MINT	0x14C5
	MERCURY_CELADON_GREEN	0x7731
	MERCURY_EMERALD	0x564F
	MERCURY_TURQUOISE_GREEN	0x4F30
	MERCURY_VERIDIAN	0x13A6
	MERCURY_HORIZON_BLUE	0xA7F9
	MERCURY_SEA_GREEN	0x2C4A
	MERCURY_MEDIUM_SEA_GREEN	0x3D8E
	MERCURY_MINT_CREAM	0xF7FF
	MERCURY_SPRING_GREEN	0x07F0
	MERCURY_PEACOCK_GREEN	0x050B
	MERCURY_MEDIUM_SPRING_GREEN	0x07D3
	MERCURY_MEDIUM_AQUAMARINE	0x6675
	MERCURY_AQUAMARINE	0x7FFA
	MERCURY_CYAN_BLUE	0x0DF1
	MERCURY_AQUA_BLUE	0x67FC
	MERCURY_TURQUOISE_BLUE	0x3739
	MERCURY_TURQUOISE	0x36B9
	MERCURY_LIGHT_SEA_GREEN	0x2595
	MERCURY_MEDIUM_TURQUOISE	0x4E99
	MERCURY_LIGHT_CYAN	0xE7FF
	MERCURY_BABY_BLUE	0xE7FF
	MERCURY_PALE_TURQUOISE	0xAF7D
	MERCURY_DARK_SLATE_GRAY	0x2A69
	MERCURY_TEAL	0x0410

	MERCURY_DARK_CYAN	0x0451
	MERCURY_CYAN	0x07FF
	MERCURY_AQUA	0xAEFC
	MERCURY_DARK_TURQUOISE	0x067A
	MERCURY_CADET_BLUE	0x5CF4
	MERCURY_PEACOCK_BLUE	0x0411
	MERCURY_POWDER_BLUE	0xB71C
	MERCURY_STRONG_BLUE	0x030E
	MERCURY_LIGHT_BLUE	0xAEDC
	MERCURY_PALE_BLUE	0x7DD9
	MERCURY_SAXE_BLUE	0x44D6
	MERCURY_DEEP_SKY_BLUE	0x05FF
	MERCURY_SKY_BLUE	0x867D
	MERCURY_LIGHT_SKY_BLUE	0x867F
	MERCURY_MARINE_BLUE	0x022F
	MERCURY_PRUSSIAN_BLUE	0x018A
	MERCURY_STEEL_BLUE	0x4416
	MERCURY_ALICE_BLUE	0xF7DF
	MERCURY_SLATE_GRAY	0x7412
	MERCURY_LIGHT_SLATE_GRAY	0x7453
	MERCURY_DODGER_BLUE	0x1C9F
	MERCURY_MINERAL_BLUE	0x0273
	MERCURY_AZURE	0x03FF
	MERCURY_WEDGWOOD_BLUE	0x5437
	MERCURY_LIGHT_STEEL_BLUE	0xB63B
	MERCURY_COBALT_BLUE	0x0235
	MERCURY_PALE_DENIM	0x5C38
	MERCURY_CORNFLOWER_BLUE	0x64BD
	MERCURY_SALVIA_BLUE	0x4C1C

	MERCURY_DARK_POWDER_BLUE	0x0193
	MERCURY_SAPPHIRE	0x092C
	MERCURY_INTERNATIONAL_KLEIN_BLUE	0x0174
	MERCURY_CERULEAN_BLUE	0x2A97
	MERCURY_ROYAL_BLUE	0x435C
	MERCURY_DARK_MINERAL_BLUE	0x21AF
	MERCURY_ULTRAMARINE	0x019F
	MERCURY_LAPIS_LAZULI	0x099F
	MERCURY_GHOST_WHITE	0xFFDF
	MERCURY_LAVENDER	0xE73F
	MERCURY_PERIWINKLE	0xCE7F
	MERCURY_MIDNIGHT_BLUE	0x18CE
	MERCURY_NAVY_BLUE	0x0010
	MERCURY_DARK_BLUE	0x0011
	MERCURY_MEDIUM_BLUE	0x0019
	MERCURY_BLUE	0x001F
	MERCURY_WISTERIA	0x5A9C
	MERCURY_DARK_SLATE_BLUE	0x49F1
	MERCURY_SLATE_BLUE	0x6AD9
	MERCURY_MEDIUM_SLATE_BLUE	0x7B5D
	MERCURY_MAUVE	0x621F
	MERCURY_LILAC	0xB4DF
	MERCURY_MEDIUM_PURPLE	0x939B
	MERCURY_AMETHYST	0x6199
	MERCURY_GRAYISH_PURPLE	0x83B4
	MERCURY_HELIOTROPE	0x5017
	MERCURY_MINERAL_VIOLET	0xBD19
	MERCURY_BLUE_VIOLET	0x895C
	MERCURY_VIOLET	0x881F

	MERCURY_INDIGO	0x4810
	MERCURY_DARK_ORCHID	0x9999
	MERCURY_DARK_VIOLET	0x901A
	MERCURY_PANSY	0x7014
	MERCURY_MALLOW	0xDA7F
	MERCURY_OPERA_MAUVE	0xE41F
	MERCURY_MEDIUM_ORCHID	0xBABA
	MERCURY_PAIL_LILAC	0xE67C
	MERCURY_THISTLE	0xDDFB
	MERCURY_CLEMATIS	0xCD19
	MERCURY_PLUM	0xDD1B
	MERCURY_LIGHT_VIOLET	0xEC1D
	MERCURY_PURPLE	0x8010
	MERCURY_DARK_MAGENTA	0x8811
	MERCURY_MAGENTA	0xF81F
	MERCURY_FUCHSIA	0xF014
	MERCURY_ORCHID	0xDB9A
	MERCURY_PEARL_PINK	0xFD9C
	MERCURY_OLD_ROSE	0xBAB3
	MERCURY_ROSE_PINK	0xFB39
	MERCURY_MEDIUM_VIOLET_RED	0xC0B0
	MERCURY_MAGENTA_ROSE	0xF874
	MERCURY_ROSE	0xF80F
	MERCURY_RUBY	0xC810
	MERCURY_CAMELLIA	0xE1D2
	MERCURY_DEEP_PINK	0xF8B2
	MERCURY_FLAMINGO	0xE457
	MERCURY_CORAL_PINK	0xFC17
	MERCURY_HOT_PINK	0xFB56

	MERCURY_BURGUNDY	0x4004
	MERCURY_SPINEL_RED	0xFB96
	MERCURY_CARMINE	0xE00B
	MERCURY_BABY_PINK	0xFEDC
	MERCURY_CARDINAL_RED	0x9806
	MERCURY_LAVENDER_BLUSH	0xFF9E
	MERCURY_PALE_VIOLET_RED	0xDB92
	MERCURY_CERISE	0xD98C
	MERCURY_SALMON_PINK	0xFC13
	MERCURY_CRIMSON	0xD8A7
	MERCURY_PINK	0xFE19
	MERCURY_LIGHT_PINK	0xFDB8
	MERCURY_SHELL_PINK	0xFD97
	MERCURY_ALIZARIN_CRIMSON	0xE126

Temperature coefficient table

{temperature， adc value}:

```
{ -5   , 2458 },
{ -4   , 2390 },
{ -3   , 2324 },
{ -2   , 2256 },
{ -1   , 2202 },
{ 0    , 2140 },
{ 1    , 2087},
{ 2    , 2025    },
{ 3    , 1965    },
{ 4    , 1909    },
{ 5    , 1854    },
{ 6    , 1797    },
{ 7    , 1748    },
{ 8    , 1695    },
{ 9    , 1642    },
{ 10   , 1594 },
```

```
{ 11   , 1543 },
{ 12   , 1499 },
{ 13   , 1451 },
{ 14   , 1406 },
{ 15   , 1363 },
{ 16   , 1320 },
{ 17   , 1279 },
{ 18   , 1240 },
{ 19   , 1191 },
{ 20   , 1165 },
{ 21   , 1127 },
{ 22   , 1093 },
{ 23   , 1057 },
{ 24   , 1024 },
{ 25   , 991      },
{ 26   , 960      },
{ 27   , 928      },
{ 28   , 900      },
{ 29   , 872      },
{ 30   , 846      },
{ 31   , 817      },
{ 32   , 790      },
{ 33   , 763      },
{ 34   , 741      },
{ 35   , 717      },
{ 36   , 697      },
{ 37   , 674      },
{ 38   , 652      },
{ 39   , 631      },
{ 40   , 611      },
{ 41   , 591      },
{ 42   , 573      },
{ 43   , 554      },
{ 44   , 536      },
{ 45   , 520      },
{ 46   , 502      },
{ 47   , 489      },
{ 48   , 476      },
{ 49   , 460      },
{ 50   , 446      },
{ 51   , 434      },
{ 52   , 419      },
{ 53   , 406   }
```