

**FerriSDK Reference Guide**

**ver. 1.1.0 (Draft)**

**Release Note**

| **Revision** | **Effect Date** | **Description of the Change** |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

**IMPORTANT NOTICE**

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH PRODUCTS OF SILICON MOTION INC (“SMI”). NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN SMI'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, SMI ASSUMES NO LIABILITY WHATSOEVER, AND SMI DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF SMI PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. SMI products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications. SMI may make changes to specifications and product descriptions at any time, without notice. SMI may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights. The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by SMI. SMI assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of SMI. Contact your local SMI sales office or your distributor to obtain the latest specifications and before placing your product order. Other names and brands may be claimed as the property of others.

**Copyright © 2015, SMI. All Rights Reserved. www.siliconmotion.com**

Contents

[1. Overview 5](#_Toc420327349)

[2. Getting Start 5](#_Toc420327350)

[2.1 File construction 5](#_Toc420327351)

[2.2 How to build sample application 5](#_Toc420327352)

[3. Update Ferri Parameter Flow 6](#_Toc420327353)

[3.1 Achieve IFerriFactory object 7](#_Toc420327354)

[3.2 Achieve IFerriDevice object 7](#_Toc420327355)

[3.3 Set MPISP 8](#_Toc420327356)

[3.4 Create ISP buffer and load from file 8](#_Toc420327357)

[3.5 Set parameters to ISP buffer 8](#_Toc420327358)

[3.6 Download ISP to Ferri and verify 8](#_Toc420327359)

[3.7 Reset Ferri 8](#_Toc420327360)

[3.8 Read ISP from Ferri and verify 8](#_Toc420327361)

[3.9 Read and verify IDENTIFY DEVICE 8](#_Toc420327362)

[3.10 Release IFerriDevice and IFerriFactory 8](#_Toc420327363)

[4. Interfaces 8](#_Toc420327364)

[4.1 APIs 8](#_Toc420327365)

[4.2 IJCInterface 9](#_Toc420327367)

[4.3 IFerriFactory 9](#_Toc420327371)

[4.4 IFerriDevice 10](#_Toc420327375)

[4.5 IIspBuffer 14](#_Toc420327393)

[4.6 Configuration ID for IIspBuffer 16](#_Toc420327407)

[4.7 17](#_Toc420327408)

# Overview

User can modify serial number, model name and other configurations to the specified value for Ferri. To modify these setting, invoking Ferri vendor command is necessary. To simplify the usage of Ferri vendor command, we provided this Ferri SDK.

Ferri SDK provides C++ interfaces. Ferri SDK supports for Microsoft Virtual C++ 2008.

# Getting Start

## File construction

-- <WORK FOLDER>

|-- *application\*

| |-- *sn\_update\_tool\*

| |-- FerriSdk\

| |-- include\

| |-- lib\

| |-- SDK\

| |-- res\

| |-- source\

| |-- test\_data\

|-- *extlib\*

| |-- vld\

|-- jcvos\

| |-- stdext\

|-- *tool\*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **File or folder name** | | **Description** | **Flag** | |
| folder | sn\_update\_tool | Sample application code for how to use FerriSdk. | M | N |
| folder | FerriSdk | Folder for FerriSdk. | M | N |
| folder | FerriSdk\include | Header files for FerriSdk. | M | N |
| folder | FerriSdk\lib | Binary static library files for FerriSdk. | M | N |
| folder | *FerriSdk\SDK* | Dynamic library for Ferri chip supporting. | M | C |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## How to build sample application

1. Unpack packaged (.rar) FerriSDK to your PC
2. Enter .\tool folder and run auto\_config.ps1
3. Enter .\application\sn\_update\_tool and open sn\_update\_tool.vcproj
4. Select config DEBUG\_STATIC\_MFC to build debug version and RELEASE\_STATIC\_MFC for release version.
5. Build solution
6. Run .\application\sn\_update\_tool\install.ps1 to create a exe package.

Install.ps1 <target folder> (for release version)

Install.ps1 <target folder> “DEBUG” (for debug version)

# Update Ferri Parameter Flow



## Achieve IFerriFactory object

Call *InitializeSdk()* to achieve *IFerriFactory* object. Returned *IFerriFactory* object is available during whole application life.

## Achieve IFerriDevice object

Call achieved *IFerriFactory* object’s *IFerriFactory::ScanDevice()* method to achieve *IFerriDevice* object. The *IFerriDevice* object attaches to the Ferri device connected to the SOLO TESTER. When all the operation completed, call *IFerriDevice::Release()* to detach Ferri device and release the object before disconnect Ferri device.

## Set MPISP

Call achieved *IFerriDevice* object’s *IFerriDevice::SetMpisp()* to pass MPISP data to *IFerriDevice* object. The MPISP should be load from binary file which is provided by SMI.

## Create ISP buffer and load from file

Call *IFerriDevice::CreateIspBuf()* to achieve *IIspBuffer* object. Then call *IIspBuffer::LoadFromFile()* to load ISP data. The ISP data is provided by SMI.

## Set parameters to ISP buffer

A series of set data and configuration methods are provided in *IIspBuffer* interface. Call necessary methods to modify Ferri’s configuration and set serial number, model name and etc. to ISP buffer. See chapter 4.5 for more information.

## Download ISP to Ferri and verify

After configuration and specified data was set to ISP data, it is necessary to download the ISP data to Ferri device. Call IFerriDevice::DownloadIsp() to download it.

After ISP data was downloaded, IFerriDevice::ReadIsp() method can be called to read ISP data from Ferri. The IIsp::Compare() method can be used to compare if the downloaded ISP data is same as the original one.

## Reset Ferri

After downloading ISP to Ferri, a reset is necessary to make new ISP run. To reset Ferri device, application can call IFerriDevice::PowerOnOff() to turn off then turn on the device power. We suggest inserting 2 seconds between power off and on.

## Read ISP from Ferri and verify

We suggest verifying downloaded ISP again followed by reset. This process is same as verify ISP after download.

## Read and verify IDENTIFY DEVICE

Read IDENTIFY DEVICE data from Ferri then check the data (serial number, etc.) is correct.

## Release IFerriDevice and IFerriFactory

Release IFerriDevice and IFerriFactory object before exit.

# Interfaces

## APIs

### bool InitializeSdk(LPCTSTR ctrl\_name, IFerriFactory \* & factory);

Call this API to initialize Ferri SDK and achieve a pointer which points to an IFerriFactory object.

**Parameters:**

***ctrl\_name*** [input]: a wide string to describe the controller name.

L“LT2244” for Ferri SATA BX serials, SM2244LT

L”SM2236” for Ferri PATA BX serials, SM2236

L”SM2246” for Ferri SATA CX serials, SM2246EN / SM2246XT

***factory*** [output]: return pointer of IFerriFactory

When call this API, factory must be set to NULL.

**Return value:**

Return true if succeed, otherwise return false.

## IJCInterface

This is the base class for all interfaces.

FerriSDK itself does not automatically try to remove an object from memory when it thinks the object is no longer being used. Instead, the object programmer must remove the unused object. The programmer determines whether an object can be removed based on a reference count.

FerriSDK uses the IJCInterface::AddRef() and IJCInterface::Release() methods to manage the reference count of interfaces on an object. The general rules for calling these methods are:

When a client receives an interface pointer from method’s reference parameter, AddRef() does not need to be called. The method has already called AddRef().

Whenever the client has finished using the interface pointer, it must call Release().

### virtual void AddRef() =0;

Increase reference counting of the object.

### virtual void Release(void) =0;

Decrease reference counting of the object.

### virtual bool QueryInterface(const char \* if\_name, IJCInterface \* &if\_ptr) =0;

This method is not supported now.

## IFerriFactory

This interface provides a capability of creating an appropriate IFerriDevice object. Application can use this interface to scan all drives which connect to SOLO TESTER on PC. Then IFerriFactory creates an appropriate IFerriDevice object for the drive.

### virtual bool ScanDevice(IFerriDevice \* & ferri) = 0;

Scan all drive connect to SOLO TESTER on the PC to find out Ferri device. After device is found, create an IFerriDevice object and let it attach to the device.

**Parameters:**

***ferri*** [output]: return pointer of IFerriDevice object

When call this API, factory must be set to NULL.

**Return value:**

Return true if succeed, otherwise return false.

### virtual bool CreateDevice(HANDLE dev, IFerriDevice \* & ferri) = 0;

Create an IFerriDevice object which related to dev. Application should open drive, and then passes the drive’s handle to this method.

The dev must be a Ferri device which is specified in InitializeSdk() API. Otherwise this method returns NULL for ferri and return false.

**Parameters:**

***dev*** [input]: handle of Ferri drive.

When application passes the device handle and the method return an object, application shall never access the handle. Application should neither close this handle. It will be closed when IFerriDevice is released.

***ferri*** [output]: return pointer of IFerriDevice object

When call this API, factory must be set to NULL.

**Return value:**

Return true if succeed, otherwise return false.

### virtual bool CreateDummyDevice(IFerriDevice \* & ferri) = 0;

This method is for debug only.

## IFerriDevice

This interface provides capabilities to access a Ferri device. Application can use this interface to read ISP from device, download ISP to device and etc.

### virtual bool CheckIsFerriChip(void) = 0;

Check if attached Ferri device matches the device for IFerriDevice object.

**Return value:**

Return true if attached device matches IFerriDevice object.

### virtual bool GetIspVersion(LPVOID isp\_ver, JCSIZE buf\_len) = 0;

Achieve firmware version from Ferri device. The returned data only used for version comparison. Application should not access the return data directly.

**Parameter:**

***isp\_ver*** [input/output]: provide a buffer pointer to achieve ISP version information.

***buf\_len*** [input]: buffer size of *isp\_ver* in byte.

**Return value:**

Return true if succeed, otherwise return false. The failure reason may be isp\_ver buffer size is not enough.

**Reference:**

*IIspBuffer::GetIspVersion()*

### virtual bool ReadFlashId(LPVOID buf, JCSIZE buf\_len) = 0;

Achieve FLASH ID of attached Ferri device and save it into buffer. The NAND flash construct information is saved in FLASH ID. Application can use FLASH ID to check if the connected Ferri is a correct one.

Application does not need to call this method directly. It is better to call CheckFlashId() to compare FLASH ID.

**Parameter:**

***buf*** [input/output]: provide a buffer pointer to achieve FLASH ID.

***buf\_len*** [input]: buffer size of *buf* in byte. For SM2246, the buffer size should be at least 1024 bytes. For other controllers, at least 512 bytes.

**Return value:**

Return true if succeed, otherwise return false. The failure reason may be *buf* buffer size is not enough.

**Reference:**

*CheckFlashId()*

### virtual bool CheckFlashId(LPVOID fid, JCSIZE len) = 0;

Compare if the FLASH ID in Ferri device is same as the data provided.

**Parameter:**

***fid*** [input]: provide a buffer pointer to achieve READ SMART data.

***len*** [input]: buffer size of buf in byte. For SM2246, the buffer size should be at least 1024 bytes. For other controllers, at least 512 bytes.

**Return value:**

Return true if succeed, otherwise return false.

### virtual bool CreateIspBuf(JCSIZE data\_len, IIspBuffer \* & isp) = 0;

Create an ISP buffer object for the attached Ferri device.

**Parameter:**

***data\_len*** [input]: This value is reserved for future use. Please set it to 0.

***isp*** [output]: a pointer to achieve IIspBuffer object. This parameter must be set to NULL when call the method.

**Return value:**

Return true if succeed, otherwise return false.

**Reference:**

*IIspBuffer::LoadFromFile()*

### virtual bool ReadIsp(JCSIZE data\_len, IIspBuffer \* & isp1, IIspBuffer \* & isp2) = 0;

Read ISP data from Ferri device. There are 2 blocks to save duplicated ISP data.

**Parameter:**

***data\_len*** [input]: The length of ISP data to read. The length can be achieved from IIspBuffer::GetSize() method by a reference ISP object.

***isp1*** [output]: an IISpBuffer object pointer to achieve ISP data from ISP block 1. This parameter must be set to NULL when call the method.

***isp2*** [output]: an IISpBuffer object pointer to achieve ISP data from ISP block 2. This parameter must be set to NULL when call the method.

**Return value:**

Return true if succeed, otherwise return false.

**Reference:**

*IIspBuffer::GetSize()*

### virtual bool DownloadIsp(IIspBuffer \* isp) = 0;

Download ISP to Ferri device.

**Parameter:**

***isp*** [input]: A pointer to IIspBuffer object. This parameter must not be NULL.

IIspBuffer object can be achieved by method *CreateIspBuf()* or *ReadIsp()*.

Before call this method, must call *SetupMpisp()* to set MPISP

**Return value:**

Return true if succeed, otherwise return false.

**Reference:**

*CreateIspBuf(),ReadIsp(), SetupMpisp()*

### virtual bool SetMpisp(LPVOID mpisp, JCSIZE mpisp\_len) = 0;

Pass MPISP data to IFerriDevice object. MPISP is internal data used for download ISP. Application should load MPISP from specified .bin file and pass the raw data to SDK.

**Parameter:**

***mpisp*** [input]: pointer to MPISP data buffer.

***mpisp\_len*** [input]: data size of *mpisp* in byte.

**Return value:**

Return true if succeed, otherwise return false.

### virtual UINT GetInitialiBadBlockCount(void) = 0;

Achieve the number of initial bad block.

**Return value:**

Return number of initial bad block.

### virtual bool ReadIdentify(LPVOID buf, JCSIZE buf\_len) = 0;

Achieve IDENTIFY DEVICE data from Ferri device.

**Parameter:**

***buf*** [input/output]: provide a buffer pointer to achieve IDENTIFY DEVICE data.

***buf\_len*** [input]: buffer size of *buf* in byte. The buffer size should be at least 512 bytes.

**Return value:**

Return true if succeed, otherwise return false.

### virtual bool ReadSmart(LPVOID buf, JCSIZE buf\_len) = 0;

Achieve READ SMART data from Ferri device.

**Parameter:**

***buf*** [input/output]: provide a buffer pointer to achieve READ SMART data.

***buf\_len*** [input]: buffer size of *buf* in byte. The buffer size should be at least 512 bytes.

**Return value:**

Return true if succeed, otherwise return false.

### virtual bool PowerOnOff(bool power\_on) = 0;

Turn Ferri’s power on or off.

**Parameter:**

***power\_on*** [input]: true: turn Ferri power on, false turn Ferri power off.

**Return value:**

Return true if succeed, otherwise return false.

### virtual void Disconnect(void) = 0;

Temporarily detach device handle to other process, for example external test application.

**Reference:**

*IFerriDevice::Connect()*

### virtual bool Connect(void) = 0;

Reconnect Ferri device after calling Disconnect().

**Return value:**

Return true if succeed, otherwise return false.

**Reference:**

*IFerriDevice::Disconnect()*

### virtual TCHAR GetDriveLetter(void) = 0;

Achieve Ferri device’s drive letter in Windows. For example when Ferri device is drive [\\.\E](file:///\\.\E):, returns ‘E’.

**Return value:**

Ferri device’s drive letter in windows.

**Reference:**

*IFerriDevice::GetDriveName()*

### virtual void GetDriveName(CJCStringT & str) = 0;

Achieve Ferri device’s drive name in windows. For example when Ferri device is drive [\\.\E](file:///\\.\E):, returns “\\.\E:”

**Parameter:**

***str*** [output]: returns drive name.

**Reference:**

*IFerriDevice::GetDriveLetter()*

### virtual UINT GetTesterPort(void) = 0;

Achieve test port ID on SOLO TESTER. This method is for internal using only.

**Return value:**

Port ID of SOLO TESTER.

## IIspBuffer

### virtual void SetCapacity(UINT cap) = 0;

Set device capacity (total sectors) to ISP buffer

**Parameter:**

***cap*** [input]: new total sectors

### virtual bool SetConfig(UINT prop\_id, UINT val) = 0;

Set value for specified configuration to ISP buffer.

**Parameter:**

***prop\_id*** [input]: configuration id to set value.

***val*** [input]: new value for specified configuration.

**Return value:**

Return false if Ferri does not support the specified configuration. Return true if succeed.

### virtual void SetSerianNumber(LPCTSTR sn, JCSIZE buf\_len) = 0;

Set serial number to ISP buffer. The serial number will be shown in IDENTIFY DEVICE. If the buffer length is shorter than the serial number size in ATA standard, only specified length will changed.

**Parameter:**

***sn*** [input]: a pointer points to serial number string buffer.

***buf\_len*** [input]: buffer size of serial number string buffer in character.

### virtual void SetVendorSpecific(LPVOID buf, JCSIZE buf\_len) = 0;

Set vendor specific to ISP buffer. The vendor specific will be shown in IDENTIFY DEVICE. If the buffer length is shorter than the vendor specific size in ATA standard, only specified length will changed.

**Parameter:**

***buf*** [input]: a pointer points to vendor specific buffer.

***buf\_len*** [input]: buffer size of vendor specific buffer in byte.

### virtual void SetModelName(LPCTSTR model, JCSIZE buf\_len) = 0;

Set model name to ISP buffer. The model name will be shown in IDENTIFY DEVICE. If the buffer length is shorter than the model name size in ATA standard, only specified length will changed.

**Parameter:**

***model*** [input]: a pointer points to model name string buffer.

***buf\_len*** [input]: buffer size of model name string buffer in character.

### virtual bool LoadFromFile(LPCTSTR fn) = 0;

Load ISP data from specified binary file.

**Parameter:**

***fn*** [input]: ISP data file name.

**Return value:**

Return true if succeed, otherwise return false.

**Reference:**

*IIspBuffer::SaveToFile()*

### virtual bool SaveToFile(LPCTSTR fn) = 0;

Save ISP data to specified binary file.

**Parameter:**

***fn*** [input]: ISP data file name.

**Return value:**

Return true if succeed, otherwise return false.

**Reference:**

*IIspBuffer::LoadFromFile()*

### virtual DWORD CheckSum(void) = 0;

Achieve checksum of ISP data.

**Return value:**

Return checksum of ISP data.

### virtual bool GetIspVersion(LPVOID buf, JCSIZE buf\_len) = 0;

Achieve firmware version in ISP buffer. The returned data only used for version comparison. Application should not access the return data directly.

**Parameter:**

***isp\_ver*** [input/output]: provide a buffer pointer to achieve ISP version information.

***buf\_len*** [input]: buffer size of *isp\_ver* in byte.

**Return value:**

Return true if succeed, otherwise return false. The failure reason may be isp\_ver buffer size is not enough.

**Reference:**

*IFerriDevice::GetIspVersion()*

### virtual bool Compare(IIspBuffer \* isp) const = 0;

Binary compare if 2 ISP data are same.

**Parameter:**

***isp*** [input]: pointer points to IIspBuffer object for comparison.

**Return value:**

Return true if 2 ISP are binary same, return false if different.

### virtual JCSIZE GetSize(void) const = 0;

Achieve data size of ISP in the object.

**Return value:**

Data size in byte.

### virtual LPVOID GetBuffer(void) = 0;

Achieve buffer pointer of ISP buffer. This method is only for internal using. Do not access the returned buffer directly.

**Return value:**

Pointer points to ISP buffer.

### virtual void ReleaseBuffer(void) = 0;

Detach buffer from IIspBuffer object. This method is only for internal using. Buffer will be freed automatically. Application does not need to call this method.

## Configuration ID for IIspBuffer

**SM2236 (PATA BX)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **ID** | **Note** | **Value** |
| FCONFIG\_HEDERS | 6 | Headers for each cylinder | 0~255 |
| FCONFIG\_SECTORS | 7 | Sectors for each header | 0~255 |
| FCONFIG\_UDMA\_LEVEL | 8 | Support UDMA level | 0~7 |

**SM2244LT (SATA BX)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **ID** | **Note** | **Value** |
| FCONFIG\_SATA\_SPEED | 1 | Max SATA speed selection. | 1: Force GEN1  2: GEN1 or GEN2 |
| FCONFIG\_TRIM | 2 | Support or not support TRIM command | 0: not support  1: support |
| FCONFIG\_DEVSLP | 3 | Support or not support device sleep feature | 0: not support  1: support |

**SM2246EN (SATA CX)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **ID** | **Note** | **Value** |
| FCONFIG\_SATA\_SPEED | 1 | Max SATA speed selection. | 1: Force GEN1  2: Force GEN1 or 2  3: GEN1, 2 or 3 |
| FCONFIG\_DEVSLP | 3 | Support or not support device sleep feature | 0: not support  1: support |

## 