# FINGERCHIP SOFTWARE DEVELOPMENT KIT (SDK) From: v14.00

This document describes the software interface used by application developers to create an application using a FingerChip device connected to a PC. It includes:

- The detailed specifications of the software interface between the application and the Dll.
- Information for driver developers

The SDK is composed of:

- this document: Document\An31_SDK.pdf
- Sdk_sources\ contains the source code of FC_Training application. Starting with this application is the convenient way to learn how to use of the Dll and the drivers.
- Delivery\: contains the files installed by the Demonstration Kit Setup.
  FC_Demo.exe, FC_Mouse.exe, FC_Parameters.exe and FC_Training.exe in <Chosen target directory>
  FingerChip.dll in <WINSYSDIR>
  An33_appli.pdf in <Chosen target directory>\Documentation
  FCGene.sys, FCUSB.sys, bioload.sys and Usbbiok.sys in <WINDIR>\System32\Drivers
  FCGene.inf, Fcusb.inf, Bioki.inf and Biokiloader.inf in <WINDIR>\Inf
  (for example, with Windows 98, <WINSYSDIR> is Windows\System, and <WINDIR> is Windows)
  Ethernet configuration file (used with AT77C104) in <Chosen target directory> (DHCP.exe)

This SDK has to be used for any new development (the release 11.xx is now abandoned). This SDK does not apply for Pocket PCs or any PDA-like embedded device. Another SDK exists, adapted to this kind of devices.

The SDK contains the driver and the DLL and the Setup sources, which also initialize FingerChip Windows's registry keys.

**Biometrics FingerChip™**

**Application Note**

**(AN-31)**

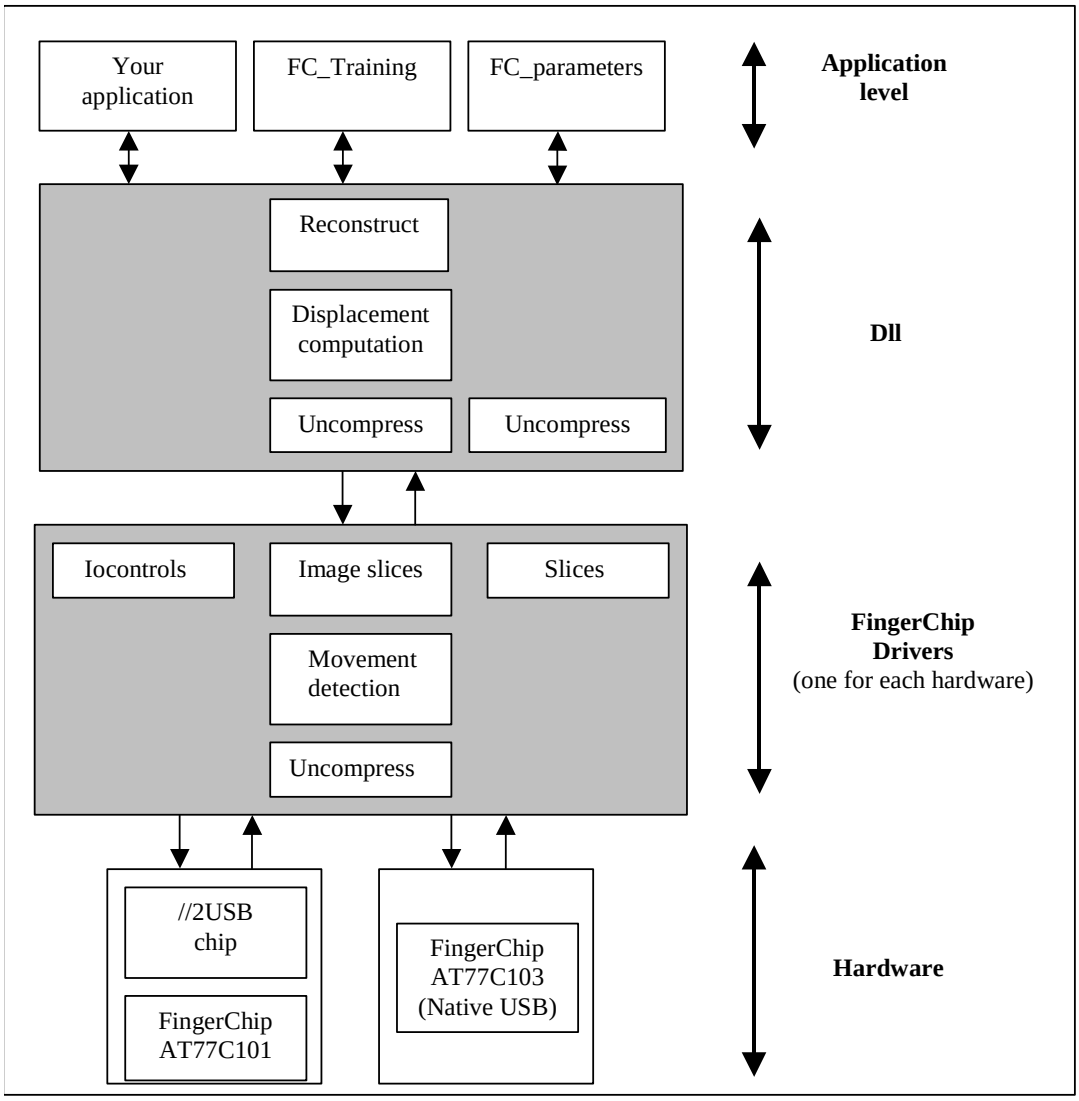
AN 31-06/03

## Primary development tool

- Visual C++ 6.0 from Microsoft

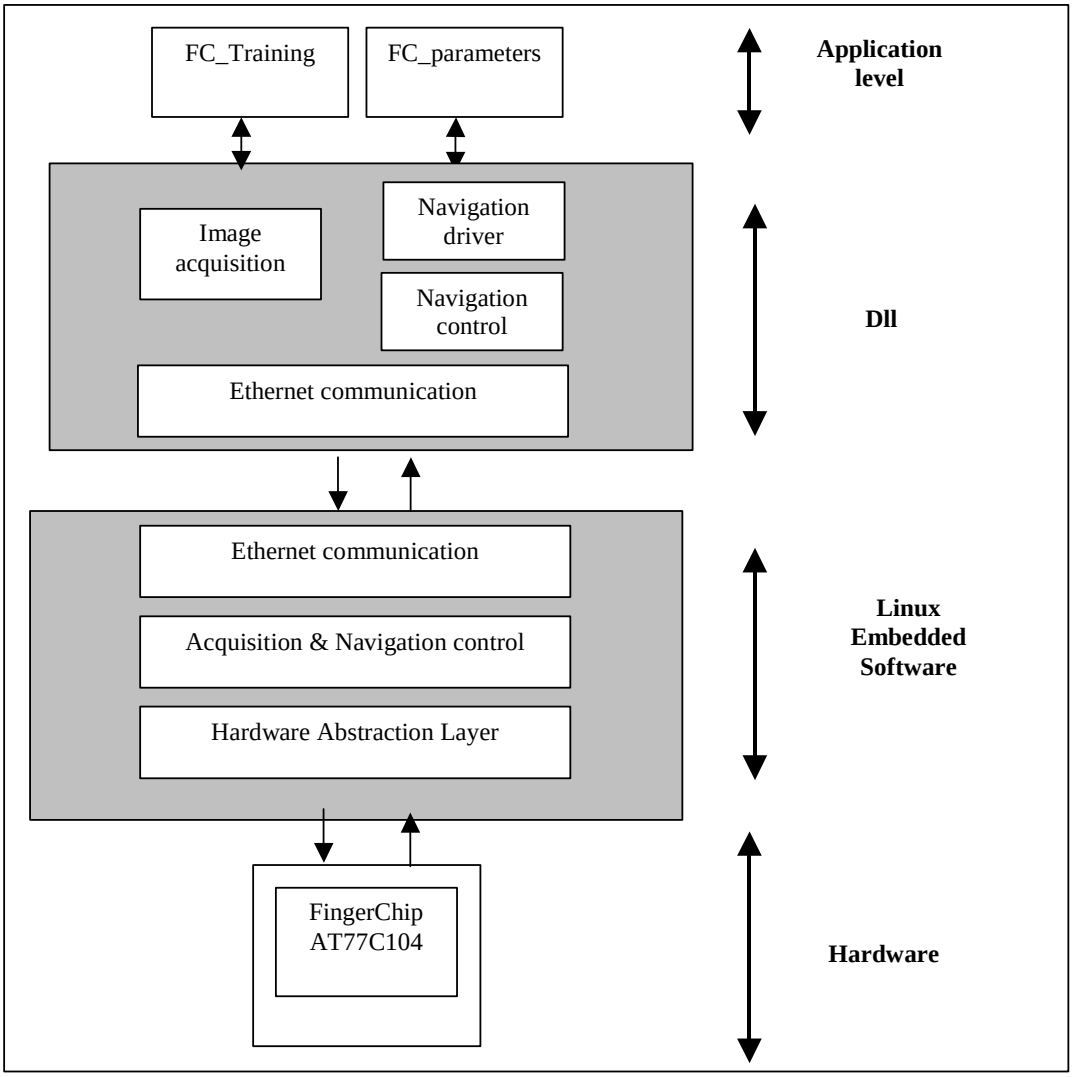
# ARCHITECTURE OVERVIEW

The following scheme shows the FingerChip system architecture, from the FingerChip silicon to the high level applications.




This architecture has been driven by cost and performance aspect. The hardware is reduced to its minimum (FingerChip and USB device) for cost reduction. The direct consequence is a complexity transfer to the software component, mainly on the driver. For performance reason, the acquisition is executed in kernel mode while the data treatment is performed in user mode, insuring a better performance for acquisition ("stop and go" during the critical acquisition time will not occur because of a user program).

For AT77C104 reader the architecture is:




Note that the Dll is fully hardware independent.
Please pay attention to these special programs:

- FC_Parameters is an application used to set default parameters of the FingerChip depending on the demonstrator's part number. It also enables the adjustment of some basic parameters that are not frequently changing such as rotation of image, temperature adjustment strategy…
- FC_Training: is a sample application that simply acquires fingerprints in a loop. This application is very useful to check if the device is working properly.

# DLL USAGE

This chapter describes how to use the Dll. It includes the user software configuration management (how and where are stored the Dll user parameter), the detailed software interface (Dll API) and the upper application architecture. This last point explains the multi-thread application architecture to implement in order to correctly handle the Dll.

## User Software Configuration

Each driver must have a key entry. Besides, the Dll has an associated key entry too.
Dll handles the user software configuration, not the application. It is under the Dll responsibility to save and restore the user configuration between two sessions.
The user software configuration is stored in the Windows registry. When the Dll is loaded, the data stored in the Windows registry are restored into shared structures (these structures are defined later). The structures are shared between the different instances of the Dll. The values of these structures are modified each time the user changes a parameter (using FC_SetSoftwareConfiguration () or FC_SetDeviceInfo ()); all the applications using this Dll immediately use the updated shared structures. At the Dll unload, the shared structures are backup into the Windows registry.
Note that there is no synchronization at runtime between the Windows registry and the shared structures in the Dll.
If the user changes the value of the parameters directly in the Windows registry, the modification will take effect at the next load of the Dll (if the software configuration backup has not overwritten the modified value). It is recommended to initialize registry keys only when the software is being installed: the sdk contains an example of registry keys initialization done with InstallShield Professional.

# API Overview

This chapter presents the API functions, with their parameters and returned status, and explains how to convert these returned error code into human readable message.
The API provides two sets of functions:

- The normal one that should be used by any application such as open/reset the device, get an image, get navigation information (for devices allowing it), close the device.
- The advanced one that enables to perform rather unusual operations such as “get some slices”, reconstruct an image from already acquired slices, output traces for debug purpose.

Note that all the functions proposed in the API are synchronous. Refer to Recommended Application Architecture paragraph for the details and the consequences of these synchronous function calls.

## Functions

First, you will need to open and close the device, using:

- FC_OpenDevice()
- FC_CloseDevice()

To obtain navigation and click information, the user will call

- FC_StartNav()
- FC_GetNav()
- FC_StopNav()

To obtain a fingerprint image, the user will call

- FC_GetImage()

The following function kills the current acquisition process. It can be called at any time in order to keep control of the system:

- FC_StopAcquisition()

If a warning or an error occurs, it is recommended to use the function

- FC_FormatMessage()

That will return an English string explaining the error. This will be very useful to the user to make the appropriate action, without having to process all the cases. In case of new error codes, this will insure to be properly processed without having to compile again the application.
If you need to get some parameter information (such as the pitch of the sensor), or change some parameters (such as mirror the image), you will use the functions to read/write the structures t_FC_DeviceInfo and t_FC_softwareConfiguration. Note that the t_FC_SliceInfo and t_FC_Imageinfo structures are returned if specified by the function that modify them.

- FC_GetSoftwareConfiguration()
- FC_SetSoftwareConfiguration()
- FC_GetDeviceInfo()
- FC_SetDeviceInfo()

The structure t_FC_DeviceInfo contains information that can not be modified if a device is active. The direct consequence is that the function FC_SetDeviceInfo() returns an error code ‘FC_Busy’. To solve this problem, you must close the device (FC_CloseDevice()) before using FC_SetDeviceInfo().
The temperature management uses the following specific routines to access the parameters. Note that a separated thread is running once the Dll is loaded to control the temperature.

- FC_GetCurrentTemperature()
- FC_SetTemperatureManagement()

For some demonstrator, the led can be controlled by software, at the Dll level, to indicate that the device is waiting for a finger or to show that the thermal regulation is active.

- FC_SetLedMode()

Some special acquisition functions are provided to just read some slices without any reconstruction, to detect if a finger is present on the FingerChip:

- FC_IsFinger()
- FC_GetSlices()

For debug purpose, a function enables to get a trace of the current process, and also to reconstruct an image with given slices (no acquisition performed):

- FC_DebugFileGeneration()
- FC_ReconstructImage()

## Structures

Some of the API functions parameters are based on structures.
Four structures are defined to set and/or obtain detailed information about the system, the image or the slices. In some cases, it is not possible to modify a data, and particularly during an acquisition.

- t_FC_DeviceInfo **:** hardware and miscellaneous information.
- t_FC_SliceInfo **:** returns information of the last acquisition
- t_FC_ImageInfo **:** returns information about the last reconstructed image
- t_FC_SoftWareConfiguration **:** set some basic parameters such as rotation, thermal
- t_FC_NavigationInfo **:** returns information of navigation and click

## Managing error codes

As mentioned earlier, each Dll function returned code can be translated into human readable string with the function FC_FormatMessage(). The error codes answer to specific rules, which define several error code ranges with equivalent seriousness. Refer to 0

Navigation information

The structure t_FC_NavigationInfo contains all the information concerning navigation.

***t_FC_NavigationeInfo***

| Parameter name | Type | Details | Update User |
|---|---|---|---|
| reserved | unsigned short | reserved | No. Update by Dll |
| reserved | unsigned short | reserved | No. Update by Dll |
| lXDisplacement | long | Horizontal displacement from the last function call | No. Update by Dll |
| lYDisplacement | long | Vertical displacement from the last function call | No. Update by Dll |
| reserved | unsigned long | reserved | No. Update by Dll |

Status Code paragraph for the implementation details.

# Detailed API description

## Functions

### Open a device

This function performs all the hardware initialization on the connected device. Besides, this function verifies the accessibility to the Dll to ensure that the Dll is not already used. This function can be called whenever the application needs to test the hardware.

| | | |
|---|---|---|
| **Prototype** | short FC_OpenDevice ( void ) | |
| **Description** | Open the device, initialize and synchronize the hardware and then check if the hardware is working: consistent test, automatic threshold detection, speed measurement, mean noise computing…. | |
| **Parameters** | None | |
| **Return Value** | Short | • FC_OK if everything is OK. See Status Code chapter or FC_error.h for a full list of errors. |

### Close a device

This function must be called when the application terminates. It releases allocated memory and allow other processes to access the device.

| | | |
|---|---|---|
| **Prototype** | short FC_CloseDevice ( void ) | |
| **Description** | Close the hardware. | |
| **Parameters** | None | |
| **Return Value** | Short | • FC_OK if done. See Status Code chapter or FC_error.h for a full list of errors |

### Get navigation information

This function must be called to enter navigation mode.

| | | |
|---|---|---|
| **Prototype** | short FC_StartNav ( ) | |
| **Description** | This function starts the collection of navigation information. | |
| **Parameters** | none | |
| **Return Value** | short | • FC_OK if done. See Status Code chapter or FC_error.h for a full list of errors |

This function must be regularly called and returns information about changes since previous call.

| | | |
|---|---|---|
| **Prototype** | short FC_GetNav ( t_FC_NavigationInfo * pNavigationInfo) | |
| **Description** | This function provides navigation information.. | |
| **Parameters** | t_FC_NavigationInfo * pNavigationInfo | Pointer to the navigation information. |
| **Return Value** | short | • FC_OK if done. See Status Code chapter or FC_error.h for a full list of errors |

This function must be called to quit navigation mode.

| | | |
|---|---|---|
| **Prototype** | short FC_StopNav ( ) | |
| **Description** | This function ends the collection of navigation information. | |
| **Parameters** | none | |
| **Return Value** | short | • FC_OK if done. See Status Code chapter or FC_error.h for a full list of errors |

### Format Message

This function returns English human readable string that explains the input status code. Use this function whenever you get a warning or an error in order to warn the user.

| | | |
|---|---|---|
| **Prototype** | short FC_FormatMessage ( short retVal,<br>unsigned char *pMessage,<br>unsigned long messageSize); | |
| **Description** | This function translates a status code to a string. | |
| **Parameters** | retVal | Return code to translate |
| | pMessage | Pointer on the message to fill |
| | messageSize | Size of the message buffer to fill (in bytes) |
| **Return Value** | short | • FC_OK<br>• FC_NOT_INITIALIZED if parameters are not correct<br>• FC_TRUNCATED_MESSAGE if the buffer is too small |

### Fingerprint image grabbing

The FC_GetImage function performs the image grabbing. This function provides an entire image of the detection. It assumes the application has allocated enough memory for the image.

| | | |
|---|---|---|
| **Prototype** | short FC_GetImage ( unsigned char *pImage,<br>unsigned short xImage,<br>unsigned short yImage,<br>t_FC_Imageinfo * pImageInfo) | |
| **Description** | This function waits for a finger (no more than the timeout parameter), acquires the slices and reconstructs an image. | |
| **Parameters** | unsigned char *pImage | Pointer to the image memory |
| | unsigned short xImage | Width in pixels of the requested image |
| | unsigned short yImage | Height in pixels of the requested image |
| | t_FC_Imageinfo * pImageInfo | Write-only structure that returns information on the acquired image. If NULL, this parameter is ignored |
| **Return Value** | short | • FC_OK if done. See Status Code chapter or FC_error.h for a full list of errors |

The pointer to the image memory must point to a correct memory area, at least the size of the requested image.
The image depth is 256 gray level per pixel (0 is black, 255 is white).
The first byte of the image memory contains the upper left pixel. Use flip/mirror/rotation parameters if you want to change the orientation.

### Stop acquisition

This function allows the application to stop the acquisition without waiting the end of the task. It can be called at any time.

| | | |
|---|---|---|
| **Prototype** | short FC_StopAcquisition ( void ); | |
| **Description** | This function allows another thread of the application to stop the acquisition before the end of the task. | |
| **Parameters** | none | |
| **Return Value** | short | • FC_OK if acquisition stopped.<br>• FC_DRIVER_NOT_FOUND if unable to connect to the driver<br>• See Status Code chapter or FC_error.h for a full list of errors |

## Exchange data between application and DLL

| | | |
|---|---|---|
| **Prototype** | Short FC_GetsoftwareConfiguration ( t_FC_SoftWareConfiguration * pDestSoftConf ) | |
| **Description** | This function fills the application structure pointed by pDestsoftConf with the current setting values. | |
| **Parameters** | pDestSoftConf | Pointer to a software configuration structure to be filled.<br>If the structure exists, the data are erased with the new one. |
| **Return Value** | Short | • FC_OK<br>• FC_NOT_INITIALIZED (pointer is NULL)<br>• FC_DLL_NOT_INITIALIZED (Dll not started properly)<br>• See Status Code chapter or FC_error.h for a full list of errors |

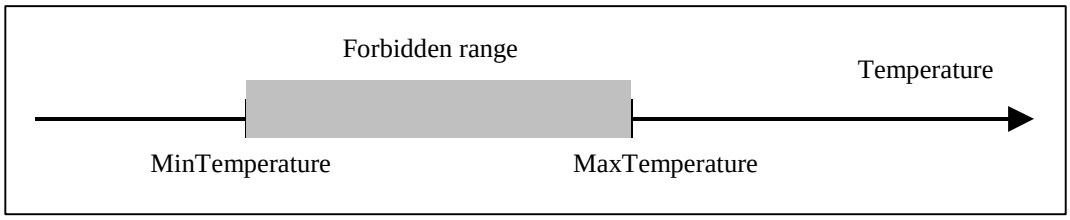
| | | |
|---|---|---|
| **Prototype** | Short FC_SetSoftwareConfiguration ( t_FC_SoftwareConfiguration * pSrcSoftConf ) | |
| **Description** | The user gives a software configuration structure that will replace the current one. | |
| **Parameters** | pSrcSoftConf | Pointer to a software configuration structure to use. |
| **Return Value** | Short | • FC_OK<br>• FC_NOT_INITIALIZED (pointer is NULL)<br>• FC_DLL_NOT_INITIALIZED (Dll not started properly)<br>• See Status Code chapter or FC_error.h for a full list of errors |

| | | |
|---|---|---|
| **Prototype** | Short FC_GetDeviceInfo ( t_FC_DeviceInfo * pDestDeviceInfo ) | |
| **Description** | This function fill the application structure pointed by pDestDeviceInfo with the current values read in the windows registry. | |
| **Parameters** | pDestDeviceInfo | Pointer to device info structure to be filled.<br>If the structure exist, the data are erased with the new one. |
| **Return Value** | Short | • FC_OK<br>• FC_NOT_INITIALIZED (pointer is NULL)<br>• FC_DLL_NOT_INITIALIZED (Dll not started properly)<br>• See Status Code chapter or FC_error.h for a full list of errors |

| | | |
|---|---|---|
| **Prototype** | Short FC_SetDeviceInfo ( t_FC_DeviceInfo * pSrcDeviceInfo ) | |
| **Description** | The user gives a device info structure that will replace the current one. | |
| **Parameters** | pSrcDeviceInfo | Pointer to device info structure to use. |
| **Return Value** | Short | • FC_OK<br>• FC_NOT_INITIALIZED (pointer is NULL)<br>• FC_DLL_NOT_INITIALIZED (Dll not started properly)<br>• See Status Code chapter or FC_error.h for a full list of errors |

## Thermal management

The temperature is a very important factor for the FingerChip as the device sense temperature differential between the finger and the sensor. If the difference is zero, then the contrast of the images becomes low. To avoid this, an on-chip temperature control element is implemented.
Several modes of operation are available, and it is possible to adjust the minimum temperature and maximum temperature of the range to avoid. Although one-degree difference enables to get nice images, as we don't know in advance the finger temperature, this range may be quite large to make sure.




When the software starts, the application must call the FC_OpenDevice function from the Dll. This function launches a separated thread in the Dll that will be responsible of the temperature management.
To set the temperature management method:

| | | |
|---|---|---|
| **Prototype** | short FC_SetTemperatureManagement ( enum eThermalManag ThermalManag ) | |
| **Description** | This function configures the thermal management. | |
| **Parameters** | ThermalManag | • THERMStart: bypass the control and always warm<br>• THERMStop: stop the control<br>• THERMSoft: automatically performed by the Dll. (default)<br>• THERMHard: reserved, not yet implemented. |
| **Return Value** | short | • FC_OK if done. See Status Code chapter or FC_error.h for a full list of errors. |

To get the current chip temperature:

| | | |
|---|---|---|
| **Prototype** | short FC_GetCurrentTemperature(long * temperature) | |
| **Description** | This function returns the current temperature of the FingerChip in Celsius. | |
| **Parameters** | short * temperature | temperature |
| **Return Value** | short | • FC_OF if properly read<br>• FC_TEMPERATURE_NOT_UPTODATE if device is in use<br>• See Status Code chapter or FC_error.h for a full list of errors |

Note that the temperature parameters are controlled in **t_FC_SoftWareConfiguration** structure: temperature unit, min and max.

If you want to implement a different strategy such as "increase of 3 degrees the temperature of the chip" to save power consumption, you will have to perform the following action:

- Stop the control if needed (THERMStop)
- Read the current temperature
- Adjust the min temperature to the current temperature – 3 degrees (to make sure), and the max temperature to the current temperature to + 3 degrees (and select Celsius or Fahrenheit!)
- Select the software control (THERMStart)
- Once the fingerprint readout is finished, stop the control (THERMStop)

As the temperature management can be achieved by a thread (if thermal policy set to software), the minimum and maximum limit for this kind of regulation can be modified in the software configuration (by FC_Parameters application for instance) and immediately take into account by the thermal management. This is also valid for a change in the thermal management.

## Led management

| | | |
|---|---|---|
| **Prototype** | short FC_SetLedMode ( enum eLedMode ledMode ) | |
| **Description** | This function configures the Led management. | |
| **Parameters** | ledMode | • LEDHard: Led is controlled by hardware<br>• LEDAuto: Dll controls the Led depending on acquisition<br>• LEDOff: Switch off the Led<br>• LEDOn: Switch on the Led |
| **Return Value** | short | • FC_OK if done. See Status Code chapter or FC_error.h for a full list of errors. |

### Finger detection

The function FC_IsFinger is used to detect the finger presence on the FingerChip. It also gives the measure of the finger displacement.

| | | |
|---|---|---|
| **Prototype** | short FC_IsFinger ( signed short *pXMove, signed short *pYMove, unsigned long * pStandardDeviation , unsigned long timeOut, unsigned long pollingValue, t_FC_SliceInfo *pSliceNfo); | |
| **Description** | Detect if a finger is moving on the FingerChip. Some slices are grabbed to compute the mean displacement. | |
| **Parameters** | signed short *pXMove | Point to the x displacement parameter. |
| | signed short *pYMove | Point to the y displacement parameter. |
| | unsigned long timeOut | TimeOut to wait before returning an error. |
| | unsigned long * pStandardDeviation | Point to a standard deviation parameter. |
| | unsigned long pollingValue | Interval time of polling in millisecond. |
| | t_FC_SliceInfo *pSliceNfo | Point to a slice info buffer. |
| **Return Value** | short | • FC_OK if finger detected.<br>• FC_NO_FINGER if finger was not detected.<br>• See Status Code chapter or FC_error.h for a full list of errors |

The finger detection is based on acquisition of several slices, at the frequency given by polling value. The pooling value is more important to get back the displacement. If the polling value is too high, the driver can miss the finger detection. The recommended value is around 20.
This function isn't available for BIOKI device.

### Slice acquisition

This function grabs a given number of slices without any reconstruction.
For BIOKI device, the number of slices can't be defined by the user (The drivers returns the amount of slices corresponding to a finger sweep)

| | | |
|---|---|---|
| **Prototype** | short FC_GetSlice ( unsigned char *pSlice, unsigned long sliceSize , t_FC_Sliceinfo * pSliceInfo) | |
| **Description** | FC_GetSlice grabs some slices and write them in the memory. | |
| **Parameters** | unsigned char *pSlice | Pointer to the slice memory |
| | unsigned long sliceSize | Size of the slice memory in bytes. If the size is not a multiple of the slice size, an entire number of slices will be returned. |
| | t_FC_Imageinfo * pImageInfo | Return structure. If NULL, this parameter is ignored. |
| **Return Value** | short | • FC_OK if done.<br>• See Status Code chapter or FC_error.h for a full list of errors |

### Debug file

The trace information allows the user to understand the function calls in the Dll and the corresponding parameters. By default, the debug file generation is disabled for security reason.

| | | |
|---|---|---|
| **Prototype** | short FC_DebugFileGeneration ( unsigned short level, char * pDebugFile) | |
| **Description** | This function changes the debug file parameters. Three levels of generation are provided :<br>▪ Disabled<br>▪ Generate debug file if an error is found<br>▪ Always generate debug File<br>When the Dll is loaded, the debug file generation is disabled. If the file exists, the trace is appended. | |
| **Parameters** | unsigned short level | • FC_LOG_DISABLED<br>• FC_LOG_ENABLE_ERROR: Generate debug file just if an error appears.<br>• FC_LOG_ENABLE_ALWAYS<br>• FC_LOG_ERASE: combined with (logical or), it allow to empty the debug file. |
| | char * pDebugFile | Path and name of the debug file (c:\temp\sweepee.log). |
| **Return Value** | short | • FC_OK if done. See Status Code chapter or FC_error.h for a full list of errors |

### Image reconstruction

This function is able to reconstruct an image from already acquired slices.

| | | |
|---|---|---|
| **Prototype** | short FC_ReconstructImage( unsigned char *pImage, unsigned short xImage, unsigned short yImage, unsigned char *pSlice, unsigned long sliceSize, t_FC_ImageInfo *pImageInfo) | |
| **Description** | This function reconstructs an image from provided slices.<br><br>The parameter pImage must point to a xImage * yImage sized buffer. The parameter pSlice must point on a sliceSize buffer. | |
| **Parameters** | unsigned char *pImage<br>unsigned short Ximage<br>unsigned short Yimage<br>unsigned char *pSlice<br>unsigned long sliceSize<br>t_FC_ImageInfo *pImageInfo | Point to the image buffer<br>x size of the image<br>y size of the image<br>point to the slice buffer<br>size of the slice buffer<br>point to a t_FC_ImageInfo structure |
| **Return Value** | short | • FC_OK if done. See Status Code chapter or FC_error.h for a full list of errors |

# Structures description

## Device characteristics

The t_FC_DeviceInfo structure contains all the hardware device information and some data concerning the Dll. The user can access those values by calling the FC_GetDeviceInfo function at any time. This structure is passed to the Dll using the functions FC_SetDeviceInfo() and FC_GetDeviceInfo().

## t_FC_DeviceInfo

| Parameter Name | Type | Details | User Update |
|---|---|---|---|
| bandwidthMode | eBandwidthMode | *Wanted bandwidth (Automatic selection, Full or Half)* | Yes |
| bytesPerSecond | double | *maximum frequency* | No. Update by Dll |
| dllVersion | char [20] | *version of Dll* | No. Compilation information |
| communicationPort | eCommPort | *Port to communicate to. (USBGenesys by default). USB, USBGenesys.* | No. Uses for debug purpose only. |
| copyright | char [300] | | No. Compilation information |
| deviceName | char [20] | *device name* | No. Deduct from 'partNumber' |
| hardVersionBoard | double | *version of Hardware board* | No. Deduct from 'partNumber' |
| maxSlicesPerSecond | double | *maximum reading speed in frame per second* | No. Update by Dll |
| partNumber | char [20] | *Device part number* | Yes. Change available if the device is not in use (not opened). |
| pixelDefaultValue | unsigned long | *Pixel default value [0..255] if other, automatic evaluation* | Yes |
| pixelPitch | unsigned short | *pixel pitch in micron (50 for FCD4B14)* | No. Deduct from 'partNumber' |
| realBandwidth | eRealBandwidth | *Obtained bandwidth (Full, Half or Nothing)* | No. Update by Dll |
| reserved | char[120] | *Reserved parameters* | Yes, but without effect |
| resolution | unsigned short | *resolution of the device* | No. Deduct from 'partNumber' |
| xSliceSize | unsigned short | *x size in pixel of one slice (280 for FCD4B14)* | No. Deduct from 'partNumber' |
| ySliceSize | unsigned short | *y size in pixel of one slice (8 for FCD4B14))* | No. Deduct from 'partNumber' |

## Slice information

The structure t_FC_SliceInfo contains all the data related to the slices, updated after each acquisition.

***t_FC_SliceInfo***

| Parameter name | Type | Details | Update User |
|---|---|---|---|
| thresholdUsed | unsigned long | *Threshold used by the Dll to detect a finger* | No. Update by Dll |
| numberOfBytesRead | unsigned long | *size of the stored slice memory, in bytes* | No. Update by Dll |
| numberOfSlicesStored | unsigned long | *number of stored slices* | No. Update by Dll |
| numberOfSlicesRead | unsigned long | *number of read slices (including waiting time)* | No. Update by Dll |
| timeToReadStoredSlices | double | *time to read the stored slices in seconds* | No. Update by Dll |
| timeToReadAllSlices | double | *time to read all the slices in seconds* | No. Update by Dll |
| slicesPerSecond | unsigned long | *acquisition speed of the current device in slices per second* | No. Update by Dll |
| currentTemperature | signed long | *current temperature* | No. Update by Dll |
| reserved | char[128] | *reserved parameters* | - |

## Image information

The structure t_FC_ImageInfo contains all the information concerning the reconstructed image.

***t_FC_ImageInfo***

| Parameter name | Type | Details | Update User |
|---|---|---|---|
| xReturnSize | unsigned long | *size in x direction of the returned image in pixel* | No. Update by Dll |
| yReturnSize | unsigned long | *size in y direction of the returned image in pixel* | No. Update by Dll |
| yMeanDisplacement | double | *mean value of displacement in y direction in pixel per slice* | No. Update by Dll |
| meanFingerSpeed | double | *mean value of finger speed in slices per second.* | No. Update by Dll |
| xReconstructedSize | unsigned long | *x size of reconstructed image ( not the returned image)* | No. Update by Dll |
| yReconstructedSize | unsigned long | *y size of reconstructed image ( not the returned image)* | No. Update by Dll |
| backgroundUsed | unsigned long | *Background pixel value used during reconstruction* | No. Update by Dll |
| lastSliceInfo | t_FC_SliceInfo | *slice info structures* | No. Update by Dll |

## Software Configuration

The structure t_FC_SoftWareConfiguration contains all the parameters concerning the software. All the parameters have a default value. This structure is passed to the Dll using the functions FC_SetSoftwareConfiguration() and FC_GetsoftwareConfiguration().

***t_FC_SoftWareConfiguration***

| Parameter name | Type | Details | User Update<br>Note: can be changed even if the semaphore is used (device in use). |
|---|---|---|---|
| rotation | char | *Image Rotation, change the sweeping way* | Yes. |
| mirror | char | *Horizontal image mirror (horizontal left <-> right)* | Yes. |
| flip | char | *Vertical image mirror (vertical up <-> down)* | Yes. |
| minHeightImage | unsigned long | *If y image* < minHeightImage *an error code is generated ( default: 100 pixels )* | Yes. |
| autoThreshold | char | *1: automatic threshold<br>0: use beginThreshold value (default: automatic)* | Yes. Deduct from 'partNumber'. |
| beginThreshold | unsigned long | *Threshold to start acquisition (default: 20)* | Yes. Deduct from 'partNumber'. |
| waitingTimeOut | unsigned long | *Timeout when waiting for a finger (default: 15 seconds)* | Yes. |
| noiseFilter | char | *Remove fixed noise or not (default: true = remove)* | Yes. |
| enhanceImage | char | *Enhance reconstructed image (default: false)* | Yes. |
| autoThermalManag | enum eThermalManag | *Thermal management :*<br>▪ *THERMSoft by software*<br>▪ *THERMStart, THERMStop*<br>▪ *THERMHard by hardware* | Yes. |
| temperatureUnit | char | *false : °C<br>true : °F* | Yes. |
| temperatureLow | signed long | *(default 77°F)* | Yes. |
| temperatureHigh | Signed long | *(default 113°F)* | Yes. |
| minSliceNumber | unsigned long | *Minimum number of slices for an image acquisition* | Yes. |
| thermalWatchdog | Unsigned char | *Not available* | No |
| ledMode | enum eLedMode | *Led management :*<br>▪ *Only LEDHard available* | No |
| Reserved | Char[123] | *reserved parameters* | - |

## Navigation information

The structure t_FC_NavigationInfo contains all the information concerning navigation.

***t_FC_NavigationeInfo***

| Parameter name | Type | Details | Update User |
|---|---|---|---|
| reserved | unsigned short | reserved | No. Update by Dll |
| reserved | unsigned short | reserved | No. Update by Dll |
| lXDisplacement | long | Horizontal displacement from the last function call | No. Update by Dll |
| lYDisplacement | long | Vertical displacement from the last function call | No. Update by Dll |
| reserved | unsigned long | reserved | No. Update by Dll |

## Status Code

All the functions of the Dll return a status code. All the possible status codes are defined in the declaration file FC_error.h, so that application developers can process these codes.
The FC_FormatMessage() returns a human readable message (in English) based on the status code: it is recommended to use this function in order to return a human comprehensive message to the user.
To ease the error processing and limit the need to compile again if new error codes occur, the following convention is used:

| Status code | Alias | |
|---|---|---|
| 0 | FC_OK | **No error, no warning**. |
| > 0 | Positive value | **Warning**<br>If it occurs during opening, it is recommended to send a message to the user to correct a potential problem.<br>If it occurs during fingerprint acquisition, it is not necessary to stop it: just send a message to the user. Anyhow, there is a resulting fingerprint image to process, even if not very good (for instance, if the warning is "sweep too fast") |
| < 0<br>at initialization | Negative value | **Error**<br>If it occurs during opening, this is probably a hard error (device not connected for instance). Display the error message so that the user can take the appropriate action (it can help for debug purpose). |
| -xxx < … < 0<br>during fingerprint acquisition | Low negative value | **Error**<br>There is no image returned, but the device is still working. A problem occurs during or after the acquisition, and it requires trying again.<br>It is recommended to display a message without stopping the acquisition, so that the user can try again.<br>For instance, the image size was too little in the vertical direction, so the Dll has generated an error. |
| -xxx | FC_OPERATION_ABORTED | The user has requested to break the acquisition. No image returned, but this is not really an error. |
| < -xxx<br>during fingerprint acquisition | Very low negative value | **Error**<br>This is a hard error (for instance, the device has been disconnected during acquisition, which is never recommended).<br>Display the message error and stop the program so that the user can correct the problem. You will probably need to close the device and re-open to make sure everything is OK. |

# Recommended Application Architecture

The Dll interface provides only synchronous entry points. This means that one Dll function call wait for the function job to be finished before get back to the application. Most of the Dll entry points have a short execution time, but some of them can lock the application for a while. The perfect example is the FC_GetImage() function that wait for a finger to get back the image. The waiting time is programmable by application and can be very long. If your application is locked on this acquisition, you will not be able to do something else and particularly to abort the finger image acquisition.

The only solution is multi-thread. On the principle, the application will create a thread in charge of the image acquisition. The main application thread can be used to achieve other tasks and for instance, to call the function FC_StopAcquisition() to cancel the current finger image acquisition. This is exactly the FC_Training application principle. The acquisition thread loops on the FC_GetImage() and then display the acquired image. When the user quit the application, the main thread executes FC_StopAcquisition() function, which unlock the current FC_GetImage() call. The acquisition thread is informed that the FC_GetImage was interrupted by its returned error code. The acquisition thread can suicide or being killed by the main thread.

This Dll characteristic prevent the use of the Visual Basic, which is mono-thread (at least lets consider it is, due to the complexity to address multi-thread in Visual Basic). A proper COM component must be developed to correctly use the Dll from Visual Basic.

# INFORMATION FOR DRIVER DEVELOPERS

This chapter describes the useful information for driver developers.
The driver is responsible of three different parts:

- Control the FingerChip
- Finger detection
- Data acquisition from the FingerChip

## Dll interactions

The Dll is responsible of the data decoding and has no real time requirement.
The Dll is completely hardware independent. It means that the same Dll can be used with different devices (different drivers). For each new device developed, the Dll is still being the same one. If functionality is modified in the Dll, it affects all the products (and it is not necessary to modify each device).
The Dll is divided into three main parts:

- The parameters
- The communication with the driver
- The reconstruction

Many applications can access to the parameters at the same time while the access to the device is reserved for one application at the same time. When the application calls the *FC_OpenDevice()* function, the Dll verifies if no one has taken the semaphore before. If the access is possible, the Dll gives the semaphore to the Dll. The semaphore is released when the application calls the *FC_CloseDevice()* function. The Dll API respects the ANSI C, but the source code of the Dll still contains some non-ANSI C part.

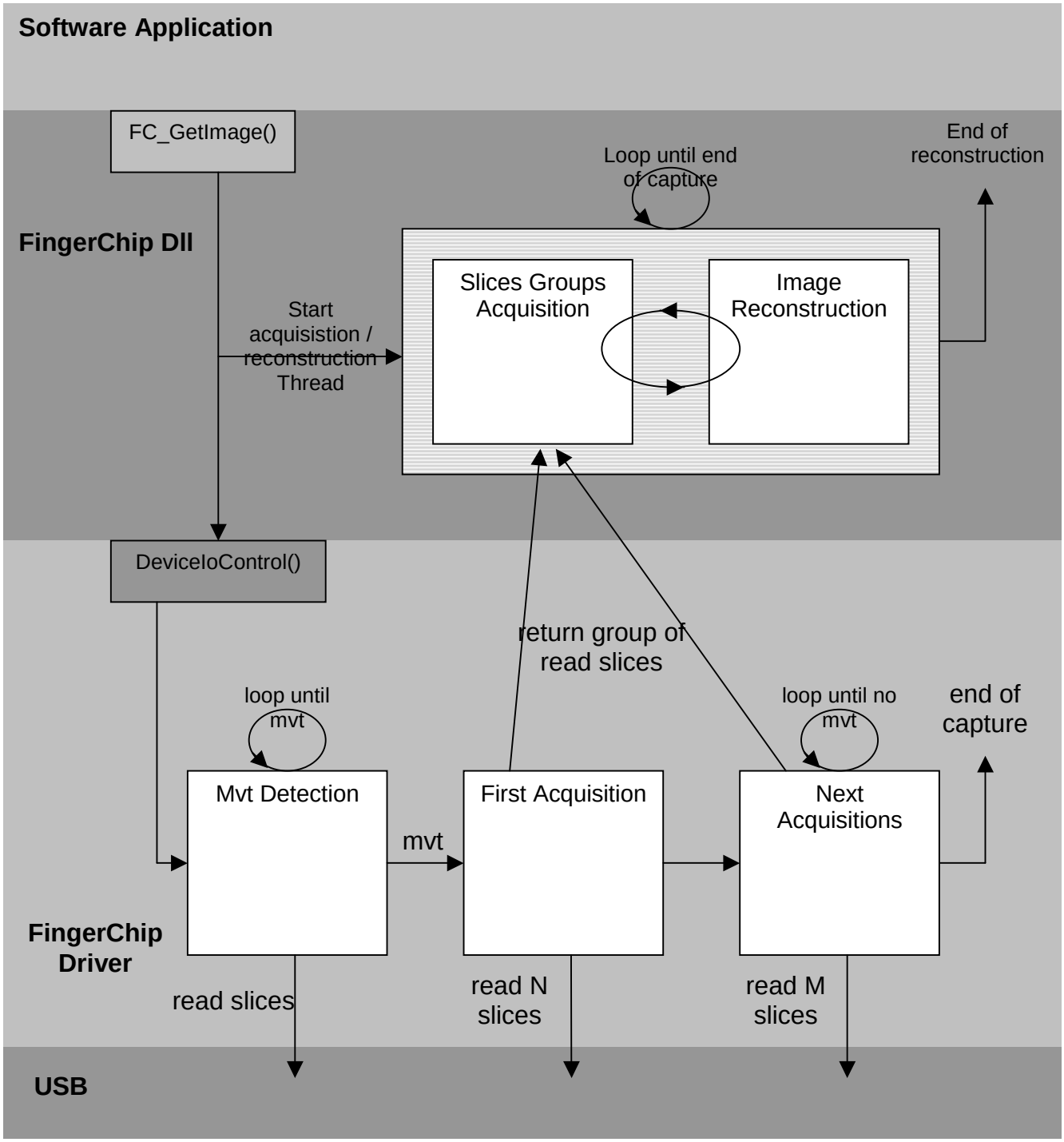
## Families of drivers

FingerChip readers are driven with two kinds of drivers:

- Drivers providing Twain data: Twain is a standard way of driving imaging devices. You can get all the documentation at www.twain.org.
- Custom drivers: the specific way they offer to drive the FingerChip readers is detailed below.

# Overview: read an image

The scheme below defines the communication between the different parts of the software during an image acquisition.



# FingerChip driver definition

For each kind of device, the driver is different but the interface with the Dll is identical.
All acquisitions are asynchronous with the Dll. It means that the Dll orders the driver to start the acquisition and the Dll is informed of the state of the driver with events.
The communication between Dll and drivers can be decomposed in three different parts: *Detection*, *First Acquisition* and *Next Acquisition*.

## Detection:

When the driver receives the FINGERCHIP_START_READING_IMAGE control codes, the main purpose of the driver is to find the finger presence and then store the next slices in the buffer.
To detect the finger presence, the driver pools the FingerChip with a defined frequency (5ms for instance) and acquires a set of slices (2 for instance). Then, the driver unpacks the data in order to compute if a finger is present. If true, the driver starts the acquisition in continuous mode (‘First acquisition phase’). If false, the pooling still executed.
The data are stored only in the acquisition phase. When the pooling is performed, the data are discarded.

## First acquisition:

When the *Detection* phase detects a finger, the *First acquisition* phase starts. The driver acquires some slices (300 for instance), then unpacks the data and computes the displacement.
Every 50 (for instance) of slices, the driver generates an event using a semaphore to inform the Dll that data are available for processing.
If a finger is still detected in the trailing slices, the driver will start the *Next acquisition* phase to keep reading the device. If the finger is not present yet, the driver generates the end event using the IoCompleteRequest API and stops the acquisition.

## Next acquisition:

The driver acquires a set of slices (150 for example), unpacks the data and detect the finger’s presence. Like the *First acquisition* phase, every 50 slices, the driver generates an event using the semaphore to inform the Dll that data are available for processing.
If no finger is detected in the trailing slices, the driver generates the end event using the IoCompleteRequest API and stops the acquisition. If the finger is still present, it loops on *Next Acquisition*.

## Stop acquisition:

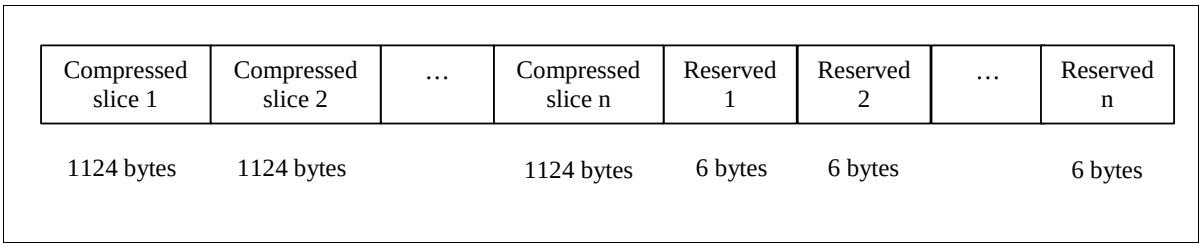
During each phase, the driver can be ordered to stop the acquisition if a FINGERCHIP_RESET_DEVICE control code is received from the Dll.
The driver must support the cancel IRP operation.

# Control code Definition

The image reconstruction is performed in the Dll. The data stored in a private buffer of the driver just contains the acquisition but the Dll has to perform the reconstruction.
The Dll is able to communicate with each driver with a set of control codes. For this reason, each driver that may be used with the Dll must implement all those functions.
To send orders to the driver, the Dll will use the "*DeviceIoControl*" functions. The different parameters to use for each control codes are described below.
When an image is requested, the Dll can provide a pointer on a semaphore in order to be informed of available slices in the buffer before the end of the acquisition. If the Dll does not need to use this mechanism, the parameter must be NULL.

## Exchanged buffers between driver and Dll

For each control codes which deal with buffer of slices, an incoming parameter takes place to define the format of the requested slices.
Today, just the COMPRESS FORMAT is allowed but this parameter may be useful for the future if the driver performs: uncompress, rotate, compute, and reconstruct. Besides, if the format of the slices sent by the hardware changes in the future, a new format can take place.
In the COMPRESS FORMAT, the slice format is:




The Compressed slice contains the 2 bytes of synchronization, the 2 bytes of temperature and the 1120 bytes of data. Each compressed slices has the 6 corresponding reserved bytes at the end of the buffer. Today in the first byte of these reserved parameters, one of the drivers writes the start of chunk and the start of chunk64.

## Read an image

| | | |
|---|---|---|
| **Control Code** | FINGERCHIP_START_READING_IMAGE | |
| **Description** | Order the driver to acquire an image. The data are stored in the buffer when a finger is detected. The acquisition stops when displacement is null on the FingerChip. | |
| **In parameters** | unsigned long timeOut | TimeOut in ms to stop finger detection |
| | HANDLE HSemaphore | Handle on the semaphore to signal when number of slices are available in the buffer. If NULL, this parameter is ignored. |
| | unsigned long pollingTime | Time in ms between a loop in the detection phase |
| | unsigned long threshold | Threshold to use in the detection phase and to detect the end of the finger |
| | unsigned long orientation | The lower bit is a copy of the rotation parameter of the Dll. The second lower bit is a copy of the mirror parameter of the Dll. The other bits are reserved. |
| | unsigned long BufferFormat | This parameter informs the driver of the requested slice buffer format. Must be COMPRESS FORMAT |
| **Out parameters** | unsigned long acquisitionTime | Elapsed time between the end and the beginning of the acquisition (not the request) in µs. |
| | unsigned char pSlices | Buffer containing the slices and the reserved bytes. |

## Read some slices

| | | |
|---|---|---|
| **Control Code** | FINGERCHIP_START_READING_SLICES | |
| **Description** | Order the driver to acquire a set of slices. The data are stored in the buffer. | |
| **In parameters** | unsigned long timeOut | TimeOut in ms to stop finger detection. Parameter ignored if WaitFinger is FALSE |
| | bool WaitFinger | If TRUE, the driver waits to find a finger before storing data and stops the acquisition when no more finger.<br>If FALSE, data are stored immediately in the buffer and the driver stops the acquisition when the buffer is full. |
| | unsigned long pollingTime | If the WaitFinger parameter is false, this parameter is ignored. Time in ms between a loop in the detection phase. |
| | unsigned long threshold | Threshold to detect the finger presence. |
| | unsigned long orientation | The lower bit is a copy of the rotation parameter of the Dll. The second lower bit is a copy of the mirror parameter of the Dll. The other bits are reserved. |
| | unsigned long BufferFormat | This parameter informs the driver of the requested slice buffer format. Must be COMPRESS FORMAT |
| **Out parameters** | unsigned long acquisitionTime | Elapsed time between the end and the beginning of the acquisition (not the request) in µs. |
| | unsigned char Slices | Buffer containing the slices and the reserved bytes. |

## Write configuration

| | | |
|---|---|---|
| **Control Code** | FINGERCHIP_WRITE_DATA | |
| **Description** | Transmit data to the FingerChip (parameter) | |
| **In parameters** | unsigned long parameter | Parameter value. |
| **Out parameters** | none | |

## Reset the device

| | |
|---|---|
| **Control Code** | FINGERCHIP_RESET_DEVICE |
| **Description** | Reset the FingerChip. |
| **In parameters** | none |
| **Out parameters** | none |

## Cancel the operation

| | |
|---|---|
| **Control Code** | FINGERCHIP_CANCEL_IO |
| **Description** | All the control codes currently running are cancelled |
| **In parameters** | none |
| **Out parameters** | none |

# Returned Status

The Dll must be informed of the driver status. When the driver accomplish a task, it sets the Error value and call the IoCompleteRequest API. When the Dll received the event, it is possible to call the GetLastError API to retrieve the status.

The possible statuses are:

- NO_ERROR
- NOT_ENOUGH_MEMORY
- INVALID_ACCESS if the device is not connected
- NOT_READY if the finger is not detected when timeOut occurs
- INSUFFICIENT_BUFFER if the buffer is full before the end of the request

# SOFTWARE SOLUTIONS

This chapter deals with some solutions proposed by ATMEL relative to specific problems.

## Low contrasted image

This problem is characteristic to the FingerChip. Its technology is based on the temperature difference detection, which became poor when the finger and the FingerChip temperature are the same.
If low contrast images or slices are obtained, check the FingerChip heating state with the Dll API function FC_GetSoftwareConfiguration(). The returned information contains the heating state: on, off or regulated (soft). On PC the recommended mode is: regulated (soft). You can switch it on by using the function FC_SetTemperatureManagement(). Then restart your acquisition.

## Logon Development – low contrasted image

A logon application based on the FingerChip ™ technology may encounter some problem relative to quality at the very first image acquisition. Actually, the heating regulation, which ensures the acquired image quality, is achieved in the Dll and can be active if an application had previously opened the device using FC_OpenDevice(). This means that the first acquisition may be low contrasted, depending of the finger and FingerChip temperature. If the FingerChip heating regulation is enable, wait a few seconds to let increase the FingerChip temperature. If the FingerChip heating is disable, refer to the chapter 0Low contrasted image.

## Logon Development – parameters management

The FingerChip parameters are defined using FC_Parameters application. They are managed by the Dll which stores them in the Windows registry.
All these keys are located in HKEY_CURRENT_USER\Software\Atmel-Grenoble\FingerChip. However this registry location isn't available at logon step. So when HKEY_CURRENT_USER isn't available, the Dll tries to use HKEY_LOCAL_MACHINE\Software\Atmel-Grenoble\FingerChip (so take care that the parameters can be saved in different locations). If both of them are not accessible the Dll can't be loaded.

# API CHANGES

## In release V14.00

New functions have been added for AT77C104A reader:

- FC_StartNav
- FC_GetNav
- FC_StopNav

## In release V13.02

The Dll API has not changed but for BIOKI01 device:

- FC_GetSlice use is different (so 'Oscilloscope' function of FC_Demo isn't available)
- FC_IsFinger isn't available (so FC_Mouse and 'Test Finger Presence And Displacement' function of FC_Demo aren't available)

## In release V13.01

The Dll API has not changed.

## In release V13.00

The Dll API has changed to support a new USB interface. In spite of these modifications, the binary compatibility of the API is preserved.

Some additional error numbers were added and the structures t_FC_DeviceInfo and t_FC_SoftwareConfiguration were modified. Developers can refer to Software Development Kit for the API details.

## In release V12.00

The DLL API has changed to ensure an ANSI C compatibility. Actually, the boolean type used in the Dll API was specific, at least, to Microsoft Visual C++ compiler. These booleans, found in the code as '**bool**', have been replaced by the **char** type. The Dll API binary compatibility is ensured (no need to recompile the application), depending on the compiler used.

- For Visual C++ 5.0 or later compiler users, the '**bool**' is implemented as a built-in type with a size of 1 byte. In a first time, the user applications do not need to be recompiled, as the Dll API binary signature is the same. But the first application recompilation will raise warnings such as "forcing char to bool : performance warning". At this stage, we recommend to change your application using the new Dll API definition.
- For Visual C++ 4.2 compiler users, the '**bool**" is mapped on an integer, using a **typedef**. As the '**int**' is 4 bytes large, your application is incompatible with the new Dll (this can cause memory corruption problems) and we highly recommend to change your application using the new Dll API definition.
- For other compiler users, make sure your applications comply with the new Dll API definition.

The modifications concern only the t_FC_SoftwareConfiguration structure.

## BUG REPORT

Send your bug reports to

fingerchip@atmel.com

with all relevant files that may help the debug.

## HISTORY

The followings show the changes compared to the previous release:

| Version | Description |
|---|---|
| OCT-2003 | Modified for v14.02 delivery |
| JUN-2003 | Modified for v14.00 delivery (including AT77C104A-EK1) |
| APR-2003 | Modified for v13.02 delivery (including BIOKI01) |
| JUL-2002 | Modified for v13.01 delivery (including FCSWEEP06) |
| MAR-2002 | Modified for v13.00 delivery |
| FEB_2002 | Delivered in a Winzip file |
| OCT-2001 | Modified for v12.00 delivery |
| SEP-2001 | Created from the internal specification 0.7.<br>Fully replaces the specification 0.7. |

## Atmel Headquarters


***Corporate Headquarters***
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

***Europe***
Atmel SarL
Route des Arsenaux 41
Casa Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

***Asia***
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

***Japan***
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581


## Atmel Product Operations


***Atmel Colorado Springs***
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

***Atmel Grenoble***
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
TEL (33) 4-7658-3000
FAX (33) 4-7658-3480

***Atmel Heilbronn***
Theresienstrasse 2
POB 3535
D-74025 Heilbronn, Germany
TEL (49) 71 31 67 25 94
FAX (49) 71 31 67 24 23

***Atmel Nantes***
La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
TEL (33) 0 2 40 18 18 18
FAX (33) 0 2 40 18 19 60

***Atmel Rousset***
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

***Atmel Smart Card ICs***
Scottish Enterprise Technology Park
East Kilbride, Scotland G75 0QR
TEL (44) 1355-357-000
FAX (44) 1355-242-743


---

***Fax-on-Demand***
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

***e-mail***
literature@atmel.com

***Web Site***
http://www.atmel.com

***BBS***
1-(408) 436-4309




AN 31-062003