ADT60 BioSIMKey

APPLICATION PROGRAMMING INTERFACE

Version 1.3   11-2005

AET60 BioCARDKey

Contents

Introduction

The ADT60/AET60 complies with the specifications defined by the BioAPI consortium (version 1.1, March 2001). ACS provides a Windows DLL layer that will make it easier for programmers to develop ADT60/AET60 applications, while still conforming to the BioAPI standards. Using this layer, programmers need not use the complicated data structures and APIs defined in the BioAPI specifications.

The ADT60/AET60 is a composite device, consisting of a fingerprint scanner and a smart card reader. This document will only cover APIs related to the fingerprint scanner. The smart card reader module will follow the PC/SC API standards.

Features

- Integrated fingerprint scanner/smart card reader

- High-speed USB interface

- Requires no additional power supply

- Portable and easily transferable from one PC to another

- High-resolution 508 DPI imaging

- Utilizes ST's patented TouchChip technology, resulting in high quality fingerprint images in any environment

- ISO 7816-3 and PC/SC compliant

- Supports all micro-controller cards, with T=0 or T=1 protocols

- Encrypted finger print template stored inside smart card

- Session key generation among smart card, ADT60/AET60 Processor and host

- Unique bonding between the smart card and ADT60/AET60

Application Programming Interface

1.1   Overview

The ADT60/AET60 layer is in the form of a Windows Dynamic Link Library (DLL) – AET60.DLL. Any programming language that can call a DLL can access this layer, such as Visual C++, Visual Basic or Delphi.

1.2   Data Types

The ADT60/AET60 layer uses the following elementary data types:

BYTE   -   8-bit unsigned value;

WORD8   -   8-bit unsigned value;

WORD16   -   16-bit unsigned value;

INT16   -   16-bit signed value;

INT32   -   32-bit signed value;

WORD32   -   32-bit unsigned value;

1.3   Data Structures

The ADT60/AET60 layer hides from the programmer almost all data structures defined by the BIOAPI specifications. It only uses 2 data structures for callback purposes.

1.3.1   AET60_DATA
This structure holds a general-purpose variable-length data.

```
typedef struct
{
        WORD32 Length; // specifies the size, in bytes, of the next element
        BYTE *Data;    // points to the data, a byte array
} AET60_DATA;
```

1.3.2   AET60_BITMAP
This structure holds a fingerprint bitmap image. If you want to develop your own callback routine, this structure will be passed as a parameter to display the fingerprint image captured by the ADT60/AET60 scanner.

```
typedef struct
{
        WORD32 Width;  // specifies the Width of the image in pixels
        WORD32 Height; // specifies the Height of the image in pixels
```

```
        AET60_DATA *Bitmap;    // points to the bitmap image
} AET60_BITMAP;
```

The Data field of the Bitmap member points to a 256-color bitmap image. Each byte in the Data field represents a pixel.

1.4 Functions

1.4.1 AET60_Open

Description

This function initializes the ADT60/AET60 library. Application must call this function first before the other ADT60/AET60 APIs.

Prototype

```
WORD32 AET60_Open ();
```

Parameters

None

Return Code

0x00000000	Success
0xFFFFFFFE	AET60_Open already called.
0xFFFFFFFF	Unable to Initialize Library. Call AET60_GetLastStatus for more details.

1.4.2 AET60_Close

Description

This function closes the ADT60/AET60 library, and frees up resources allocated by AET60_Open. Call this function when you no longer need the ADT60/AET60.

Prototype

```
WORD32 AET60_Close ();
```

Parameters

None

Return Code

0x00000000	Success
0xFFFFFFFF	AET60_Open was not called previously.

1.4.3 AET60_GetNumDevices

Description

This function returns the number of biometric devices installed in the system.

Prototype

```
INT32 AET60_GetNumDevices ();
```

Parameters

       None

Return Code

>= 0	Success. This indicates the number of BioAPI-compliant devices installed in your system.
0xFFFFFFFF	AET60_Open not called.
< 0	Call AET60_GetLastStatus for more details.

1.4.4 AET60_GetDeviceName

Description

This function returns the name of the indexed biometric device installed in the system.

Prototype

```
WORD32 AET60_GetDeviceName (INT32 index,
                           char *Name,
                           int *NameLength);
```

Parameters

[IN] index	The index number of the target biometric device. Valid values for this parameter ranges from 0 to N-1, where N is the return value of AET60_GetNumDevices.
[OUT] Name	On success, the function will fill up a NULL-terminated string into this parameter
[IN/OUT] NameLength	On entry, this parameter indicates the length (in bytes) of the Name parameter.
	On exit, the function stores the length of the Name string.
	If NameLength on entry is < NameLength on exit, the Name parameter will not be modified.

Return Code

0x00000000	Success.
0xFFFFFFFF	AET60_Open not called.
< 0	Call AET60_GetLastStatus for more details.

1.4.5 AET60_LockReader

Description

This function prepares the indexed biometric device for exclusive use for the application. You must call this function first before using the fingerprint scanner.

Prototype

```
WORD32 AET60_LockReader (INT32 index);
```

Parameters

[IN] index	The index number of the target biometric device. Valid values for this parameter ranges from 0 to N-1, where N is the return value of AET60_GetNumDevices.

Return Code

0x00000000	Success. ADT60/AET60 device is ready.
0xFFFFFFFF	AET60_Open not called.
< 0	Call AET60_GetLastStatus for more details.

1.4.6 AET60_UnlockReader
Description

This function frees the biometric device previously locked by AET60_LockReader.

Prototype

```
WORD32 AET60_UnlockReader ();
```

Parameters

None.

Return Code

0x00000000	Success.
0xFFFFFFFF	AET60_Open not called.
< 0	Call AET60_GetLastStatus for more details.

1.4.7 AET60_Capture
Description

This function scans for a live finger in the ADT60/AET60 scanner. It then returns the corresponding fingerprint template of the captured finger. You can call this function only after a successful call to AET60_LockReader.

Prototype

```
WORD32 AET60_Capture (BYTE *Template,

                      DWORD *TemplateLength,

                      DWORD TimeOut);
```

Parameters

[OUT] Template	Points to a buffer that will store the captured fingerprint template.

[IN/OUT] TemplateLength	On entry, this parameter indicates the length of the buffer pointed by Template.
	On exit, this parameter will hold the captured fingerprint template size – in bytes.
	If TemplateLength on exit is > TemplateLength on entry, the Template parameter will **not** hold the captured fingerprint Template. You will have to allocate a bigger template and call the function again.
[IN] TimeOut	Specifies how long – in milliseconds - the ADT60/AET60 will wait for a fingerprint capture.

Return Code

0x00000000	Success.
0xFFFFFFFF	AET60_Open not called.
< 0	Call AET60_GetLastStatus for more details.

1.4.8 AET60_Enroll

Description

This function will prompt user to present finger onto the scanner and consolidates the template for enrollment purpose. It then returns the corresponding fingerprint template of the captured finger. You can call this function only after a successful call to AET60_LockReader.

Prototype

```
WORD32 AET60_Enroll (BYTE *Template,

                     DWORD *TemplateLength,

                     DWORD TimeOut);
```

Parameters

[OUT] Template	Points to a buffer that will store the captured-and-consolidated fingerprint template.
[IN/OUT] TemplateLength	On entry, this parameter indicates the length of the buffer pointed by Template.
	On exit, this parameter will hold the captured fingerprint template size – in bytes.
	If TemplateLength on exit is > TemplateLength on entry, the Template parameter will **not** hold the captured fingerprint Template. You will have to allocate a bigger Template and call this function again.
[IN] TimeOut	Specifies how long – in milliseconds - the ADT60/AET60 will wait for a fingerprint capture.

Return Code

0x00000000	Success.
0xFFFFFFFF	AET60_Open not called.
< 0	Call AET60_GetLastStatus for more details.

1.4.9 AET60_Verify

Description

This function will prompt the user to present finger onto the scanner and compares it to the given fingerprint template. It will match the live finger with the given template and report the result. You can call this function only after a successful call to AET60_LockReader.

Prototype

```
WORD32 AET60_Verify  (BYTE *Template,

                      DWORD TemplateLength,

                      DWORD TimeOut,

                      BOOL *Result,

                      DWORD *Far, DWORD *Frr,

                      DWORD *AchievedFar, DWORD *AchievedFrr);
```

Parameters

[IN] Template	Points to a fingerprint template that will be matched against the scanned finger in ADT60/AET60.
[IN] TemplateLength	This parameter indicates the length of the buffer pointed by Template.
[IN] TimeOut	Specifies how long – in milliseconds - the ADT60/AET60 will wait for a fingerprint capture.
[OUT] Result	This parameter will hold the result of the matching test done by the AET60 DLL. If it is non-zero, then the finger matches the template.
[IN/OUT] Far	On entry, specifies the desired FAR scanning level. On exit, indicates the FAR scanning level used.
[IN/OUT] Frr	On entry, specifies the desired FRR scanning level. On exit, indicates the FRR scanning level used.
[OUT] AchievedFar	Indicates the FAR scanning level achieved.
[OUT] AchievedFrr	Indicates the FRR scanning level achieved.

Return Code

0x00000000	Success.
0xFFFFFFFF	AET60_Open not called.
0xFFFFFFFC	Template size is too small; or Template is invalid.
0xFFFFFFFD	Internal memory error.
< 0	Call AET60_GetLastStatus for more details.

1.4.10 AET60_Match

Description

This function will match 2 given fingerprint templates and report the result to the application. You can call this function only after a successful call to AET60_LockReader.

Prototype

```
WORD32 AET60_Match    (BYTE *Template1,

                        DWORD Template1Length,

                        BYTE *Template2,

                        DWORD Template2Length,

                        BOOL *Result,

                        DWORD *Far, DWORD *Frr,

                        DWORD *AchievedFar, DWORD *AchievedFrr);
```

Parameters

[IN] Template1	Points to the 1st fingerprint template that will be matched.
[IN] Template1Length	This parameter indicates the length of the buffer pointed by Template1.
[IN] Template2	Points to the 2nd fingerprint template that will be matched.
[IN] Template2Length	This parameter indicates the length of the buffer pointed by Template2.
[OUT] Result	This parameter will hold the result of the matching test done by the AET60 DLL. If it is non-zero, then the finger matches the template.
[IN/OUT] Far	On entry, specifies the desired FAR scanning level. On exit, indicates the FAR scanning level used.
[IN/OUT] Frr	On entry, specifies the desired FRR scanning level. On exit, indicates the FRR scanning level used.
[OUT] AchievedFar	Indicates the FAR scanning level achieved.
[OUT] AchievedFrr	Indicates the FRR scanning level achieved.

Return Code

0x00000000	Success.
0xFFFFFFFF	AET60_Open not called.
0xFFFFFFFC	Template(s) size is too small; Template(s) is invalid.
0xFFFFFFFD	Internal memory error.
< 0	Call AET60_GetLastStatus for more details.

1.4.11 AET60_GetLastStatus

Description

This function will return the BIOAPI status code of the last AET60 API called. This function is used for compatibility with BioAPI specifications.

Prototype

```
WORD32 AET60_GetLastStatus ();
```

Parameters

None.

Return Code

Please refer to Appendix A for the meaning of the status code.

1.4.12 AET60_GetStatus

Description

This function will return a null-terminated string corresponding to the BioAPI status code passed.

Prototype

```
WORD32 AET60_GetStatus (DWORD Status, char *string);
```

Parameters

[IN] Status	Status code returned by BioAPI driver.
[OUT] string	This parameter will hold the null terminated string corresponding to Status.

Return Code

Always returns 0x00.

1.4.13 AET60_GetMessage

Description

This function will return a null-terminated string corresponding to the BioAPI callback message code passed.

Prototype

```
WORD32 AET60_GetMessage (DWORD Message, char *string);
```

Parameters

[IN] Message	Message code returned by BioAPI driver.
[OUT] string	This parameter will hold the null terminated string corresponding to Message.

Return Code

Always returns 0x00.

1.4.14 AET60_SetCallback

Description

This function will set your callback routine to handle events during finger scanning. It allows your program to control the GUI while finger scanning is in progress. You can call this function only after a successful call to AET60_LockReader.

Prototype

```
int AET60_SetCallback (AET60_CALLBACK_TYPE Callback)
```

Parameters

[IN] Callback	Points to a callback function that will handle events during finger scanning. Please refer to section 3.5 for more details.

Return Code

0x00000000	Success.
< 0	Call AET60_GetLastStatus for more details.

1.5 FAR and FRR

FAR (False Acceptance Rate) is the probability that two different fingerprint data are falsely matched. FRR (False Rejection Rate) is the probability that two fingerprint data representing the same sample base do not match. Compliant to the BioAPI specifications, the user is given the option to provide the desired FAR and FRR values consistent to his program requirements.

FAR and FRR parameters are 32-bit integer values that are translated into their respective FAR and FRR values by the formula:

$$\text{FAR/FRR} = N/(2^{31}-1) \quad \text{where N = FAR/FRR parameter value}$$

The ADT60/AET60 implements five (5) security levels based on the desired FAR/FRR values defined by the user, summarized as follows:

Security	FAR		FRR	
Level	Parameter	Value	Parameter	Value
0 (Highest)	2148	0.000001	654983	0.001902
1	310419	0.000145	500364	0.000233
2	500364	0.000233	464931	0.000217
3	2515777	0.001172	358630	0.000167
4	4083440	0.001902	346819	0.000162

1.6 Callback Routine

During finger scanning (AET60_Enroll, AET60_Verify, AET60_Capture), your application will lose control as the ADT60/AET60 driver takes over. The ADT60/AET60 driver however allows your application to intercept events during scanning via the callback routine.

You can define a customized callback routine that knows what events are happening during finger scanning. The ADT60/AET60 driver will call your callback routine based on the events happening during finger scanning. The callback prototype is:

```
typedef DWORD (*AET60_CALLBACK_TYPE)

        (void         *Param,

        WORD32        GuiState,

        BYTE          Response,

        WORD32        Message,
```

```
        BYTE            Progress,
        AET60_BITMAP    *SampleBuffer);
```

Parameters

[IN] Param	Not used.
[IN] GUIState	Bit-mapped value indicating which parameter(s) is valid.
[IN] Response	Not used.
[IN] Message	Message Code that prompts user what action to take.
[IN] Progress	Not used.
[IN] SampleBuffer	Points to a bitmap image of the finger currently on the scanner.

Typical Callback Code:

Below is a typical application callback routine that handles the events sent by the AET60 driver to your application.

```
DWORD MyCallback (void *GuiStateCallbackCtx,
        WORD32 GuiState,
        BYTE Response,
        WORD32 Message,
        BYTE Progress,
        AET60_BITMAP *SampleBuffer)
{
        // check GuiState's bit if Message parameter is valid.
        if (GuiState & BioAPI_MESSAGE_PROVIDED)
        {       char string [128];
                // call AET60_GetMessage to convert Message code to English text
                AET60_GetMessage (Message, string);
                // then display it in the GUI
                DisplayString (string);
        }
        // check GuiState's bit if SampleBuffer parameter is valid.
        if (GuiState & BioAPI_SAMPLE_AVAILABLE)
        {
                // display the fingerprint image to GUI
                // display as 256-color DIB image
                DisplayDIB (SampleBuffer->Bitmap->Data, SampleBuffer->Width,
                SampleBuffer->Height);
        }
        return 0;
}
```

Appendix A: Status Codes

If the ADT60/AET60 layer returns an error code, you can check what specific BIOAPI error it has encountered by calling the following APIs:

```
AET60_GetStatus (AET60_GetLastStatus (), String);
```

Your String variable will then hold the error status last encountered. The following are a list of Status Codes returned by the BIOAPI driver. These are defined in the BioAPI header file: **bioapi_err.h**.

BioAPI_OK	OK
BioAPI_UNSUPPORTED_BIR_HANDLE	Unsupported BIR handle
BioAPIERR_BSP_ACTION_CANCELLED_BY_USER	Action cancelled by user
BioAPIERR_BSP_AUDIT_DATA_ENCRYPTION_ERROR	Audit data encryption failure
BioAPIERR_BSP_INVALID_BIR	Invalid BIR
BioAPIERR_BSP_PM_CONSOLIDATION_FAIL	Consolidation failure
BioAPIERR_BSP_PM_INIT_FPFLTR_FAIL	fpfltrInit failure
BioAPIERR_BSP_PM_INIT_MATCHER_FAIL	InitMatcher failure
BioAPIERR_BSP_PM_INIT_QUALITY_FAIL	InitQuality failure
BioAPIERR_BSP_PM_MATCH_FAIL	Match failure
BioAPIERR_BSP_PM_SET_SECURITY_LEVEL_FAIL	SetSecurityLevel failure
BioAPIERR_BSP_PP_BAD_DEVICE	Bad device
BioAPIERR_BSP_PP_BAD_STATE	PP bad state
BioAPIERR_BSP_PP_BADPARAMETER	Bad parameter
BioAPIERR_BSP_PP_BUFFER_TOO_SMALL	Buffer too small
BioAPIERR_BSP_PP_DEVICEINUSE	Device in use
BioAPIERR_BSP_PP_ERROR	PP Error
BioAPIERR_BSP_PP_INVALID_LICENSE	Invalid license
BioAPIERR_BSP_PP_NODEVICE	No device
BioAPIERR_BSP_PP_NOT_SUPPORTED	Not supported
BioAPIERR_BSP_PP_NOTAUTHORIZED	Not authorized
BioAPIERR_BSP_PP_SENSOR_COMMUNICATION	Sensor communication failure
BioAPIERR_BSP_PURPOSE_NOT_SUPPORTED	Purpose not supported
BioAPIERR_BSP_TIMEOUT_EXPIRED	Timeout expired

Appendix B: Callback Message codes

List of BIOAPI Callback message codes. These are defined in BIOAPI header file
bioapi_touchchip_bsp_gui_message.h.

TOUCHCHIP_GUI_MESSAGE_BAD_QUALITY	Bad quality
TOUCHCHIP_GUI_MESSAGE_CLEAN_SENSOR	Clean the sensor
TOUCHCHIP_GUI_MESSAGE_CONSOLIDATION_FAIL	Consolidation fails
TOUCHCHIP_GUI_MESSAGE_CONSOLIDATION_SUCCEED	Consolidation succeeds
TOUCHCHIP_GUI_MESSAGE_ENROLL_START	Enrollment GUI start
TOUCHCHIP_GUI_MESSAGE_FINGER_DETECT_START	Finger detect GUI start
TOUCHCHIP_GUI_MESSAGE_GOOD_IMAGE	Good image
TOUCHCHIP_GUI_MESSAGE_GUI_FINISH	Non-specific GUI end
TOUCHCHIP_GUI_MESSAGE_GUI_FINISH_FAIL	Operation Failed GUI end
TOUCHCHIP_GUI_MESSAGE_GUI_FINISH_SUCCEED	Operation succeeded GUI end
TOUCHCHIP_GUI_MESSAGE_KEEP_FINGER	Keep the finger
TOUCHCHIP_GUI_MESSAGE_NO_FINGER	No finger present
TOUCHCHIP_GUI_MESSAGE_PUT_FINGER	Put your finger
TOUCHCHIP_GUI_MESSAGE_PUT_FINGER2	Put finger for the 2nd time
TOUCHCHIP_GUI_MESSAGE_PUT_FINGER3	Put finger for the 3rd time
TOUCHCHIP_GUI_MESSAGE_REMOVE_FINGER	Remove finger
TOUCHCHIP_GUI_MESSAGE_START	Non-specific GUI start
TOUCHCHIP_GUI_MESSAGE_TOO_DARK	Use less pressure
TOUCHCHIP_GUI_MESSAGE_TOO_DRY	Moisten finger
TOUCHCHIP_GUI_MESSAGE_TOO_HIGH	Move down
TOUCHCHIP_GUI_MESSAGE_TOO_LEFT	Move right
TOUCHCHIP_GUI_MESSAGE_TOO_LIGHT	Press harder
TOUCHCHIP_GUI_MESSAGE_TOO_LOW	Move up
TOUCHCHIP_GUI_MESSAGE_TOO_RIGHT	Move left
TOUCHCHIP_GUI_MESSAGE_TOO_SMALL	Press whole finger
TOUCHCHIP_GUI_MESSAGE_TOO_STRANGE	Is fingerprint?
TOUCHCHIP_GUI_MESSAGE_VERIFY_START	Verification GUI start