INTERNATIONAL STANDARD

ISO/IEC 29164

First edition
2011-10-01

Information technology — Biometrics — Embedded BioAPI

Technologies de l'information — Biométrie — BioAPI incorporé

Reference number
ISO/IEC 29164:2011(E)

COPYRIGHT PROTECTED DOCUMENT

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 29164 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 37, *Biometrics*.

Introduction

The environment for embedded systems differs in many ways from that of a more general computing environment. One difference is that the amount of processing power and/or memory/storage can be more limited in the embedded environment and operating system support and resources can also be more constrained. As a result, implementation of more general purpose interfaces might not be appropriate. In the case of embedded biometric technology, the algorithms and sensors are frequently packaged into hardware/firmware modules.

It can also be the case that the designer of the embedded system is not concerned with details of the biometric technology within its software and firmware and prefers to just integrate an external module that deals with some or all biometric functionalities.

This International Standard is not meant for applications where the integration of biometric functionality is going to be done within the software or firmware of the application. In such cases BioAPI (ISO/IEC 19784-1) is to be used, or its Frameworkless version (see ISO/IEC 19784-1 with Amd.2).

The interface defined in this International Standard provides a direct connection with such biometric modules. The definition of this interface is given by the services to be provided, as well as the message formats for commands to be sent to biometric modules and responses expected from them.

This International Standard is intended to provide a common interface for all those biometric systems where BioAPI (ISO/IEC 19784-1) cannot be implemented. From the historical point of view, as BioAPI does imply relatively large requirements both in processing power and memory capacity, some different approaches have been developed. One of those approaches is the use of BioAPI without the need of using the BioAPI framework, which is one of the most consuming parts of BioAPI. That version is called Framework free BioAPI, and is standardized in the 2nd Amendment to BioAPI. But even that approach, which can be of great help for several applications, such as Biometric Applets or Biometric services in mobile devices which run an Operating System, can be too demanding for embedded systems. Therefore a new approach is standardized in this International Standard, under the name of Embedded BioAPI, which should never be confused with the Framework free version of BioAPI.

Examples of applications where Embedded BioAPI might be used include remote controls, garage door openers, auto ignitions, physical access devices, memory sticks, authentication tokens, and handheld weapons. The utility of a standard interface in this environment is less obvious than for more general purpose processing environments, but addresses two important situations:

— It allows a device (unit into which the data capture device is embedded, e.g. a remote control device) manufacturer to use the same code base for multiple devices/units in his product line that differ only in embedded data capture device/biometric technology (e.g. Device A comes with a built-in fingerprint data capture device/algorithm and Device B comes with a built-in facial recognition camera/capability). This is a configuration management (CM) and efficiency issue (the single code base simplifying CM).

— It allows an OEM data capture device manufacturer who wants to build a single OEM unit/firmware to support multiple device vendors (the same firmware regardless of what device the data capture device unit is embedded within).

Throughout the text of this International Standard, devices suitable to be using Embedded BioAPI will be referred as “Embedded BioAPI subcomponents”. Noting that other kind of devices can also use this International Standard, this notation has been used for improving understanding of the standard. This International Standard does not state any requirement for those devices (e.g. Embedded BioAPI subcomponents) but those needed as to implement Embedded BioAPI.

Information technology — Biometrics — Embedded BioAPI

1 Scope

This International Standard provides a standard interface to hardware biometric modules designed to be integrated in embedded systems which can be constrained in memory and computational power. This International Standard specifies a full interface for such hardware-based biometric modules. This interface, called Embedded BioAPI, is defined by the specification of commands to be implemented by these modules. Such a specification is done in two levels:

— For low level implementations, a frame definition is provided, as well as the coding of all commands and their relevant responses. Being defined as a single-master/multiple-slave half-duplex protocol, these messages can be implemented over any communication interface at the physical and link layers. The definition of such communication interfaces is outside of the scope of this International Standard.

— A C-based function header description, for those manufacturers that want to provide a C-library for integration as a Software Development Kit for the overall embedded system.

Regarding security, this International Standard defines two kinds of devices:

— Type A: devices that, due to lack of processing capabilities, do not implement any kind of security mechanism.

— Type B: devices that implement security mechanisms for achieving confidentiality, integrity and/or authenticity. Use of the Type B kind of devices is recommended. For Type B devices a set of minimum requirements is defined, but the security mechanisms to be used are out of the scope of this International Standard.

Low level implementation is outside of the scope of the normative part of this International Standard, although an informative annex (see Annex B) is provided.

Security mechanisms, although considered in this International Standard, are outside of the scope of this International Standard, and are referred to other relevant standards. In particular, key management is outside of the scope of this International Standard, and is expected to be done prior to the application of this International Standard.

Specifications and requirements for Embedded BioAPI subcomponents, or any kind of devices suitable to implement Embedded BioAPI, are outside of the scope of this International Standard.

2 Conformance

A biometric module conforms to this International Standard by covering all mandatory items in the normative parts. A biometric module conformant to this International Standard can provide additional functionality as long as it does not modify the behaviour stated in this International Standard.

A more detailed list of all conformance requirements can be found in Annex A.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 19784-1:2006, *Information Technology — Biometric application programming interface — Part 1: BioAPI specification*

ISO/IEC 19784-1/Amd.3:2010, *Information technology — Biometric application programming interface — Part 1: BioAPI specification — Amendment 3: Support for interchange of certificates and security assertions, and other security aspects*

ISO/IEC 19785-1:2006, *Information Technology — Common Biometric Exchange Formats Framework — Part 1: Data Element Specification*

ISO/IEC 19785-3:2007, *Information Technology — Common Biometric Exchange Formats Framework — Part 3: Patron format specifications*

ISO/IEC 19794 (all parts), *Information Technology — Biometric data interchange formats*

ISO/IEC 24761:2009, *Information technology — Security techniques — Authentication context for biometrics*

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

NOTE Function names and data element names are not included here, but are defined within the body of this International Standard.

4.1
biometric module
hardware-based module that implements some or all biometric functions related to a biometric modality, i.e. capture, sample processing, comparison, storage, enrolment, or any logical combination of them

NOTE The Biometric module might provide other functionalities, such as sending signals for the activation of external services, but such functionality is outside of the scope of this International Standard.

4.2
biometric sample
information obtained from a biometric sensor, either directly or after further processing

NOTE See also raw biometric sample, intermediate biometric sample, and processed biometric sample in ISO/IEC 19784-1:2006.

4.3
biometric template
biometric sample or combination of biometric samples that is suitable for storage as a reference for future comparison

4.4
Embedded BioAPI subcomponent
subcomponent provided to system integrators for integration into a more complex system or device

NOTE 1 Subcomponents might be provided by third-parties or the manufacturer itself.

NOTE 2 This International Standard does not state any requirement for such subcomponents, but those needed to implement Embedded BioAPI.

4.5
embedded system
special-purpose computer system designed to perform one or a few dedicated functions, sometimes with real-time computing constraints

NOTE It is usually embedded as part of a complete device including hardware, firmware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming.

4.6
frame
set of bytes that conform a command or a response message within a communication between two devices

4.7
general processing unit
element in a digital system in charge of the control of part or all of the information processing, which is usually a microprocessor, microcontroller or a microprocessor-based subsystem

4.8
host
processing unit of the embedded system that is to be directly connected to the biometric modules

4.9
Master Embedded BioAPI interface
subset of Embedded BioAPI based on sending commands and accepting responses

4.10
processed biometric sample
biometric sample suitable for comparison

4.11
session key
cryptographic key for ciphering and/or authenticating, that changes after each use within a session

NOTE A session can be defined as the time between power on and power off, as the time between a device attach function and a device detach function, etc. After detecting a failure in complying with the security mechanisms defined, the session shall be considered as expired.

4.12
Slave Embedded BioAPI interface
subset of Embedded BioAPI based on accepting commands and sending responses

4.13
template
biometric template

4.14
template storage capacity
amount of biometric template BIRs that can be stored in a biometric module

5 Symbols and abbreviated terms

BIR – Biometric Information Record

OEM – Original Equipment Manufacturer

PoS – Point of Service terminal

RFU – Reserved for Future Use

SDK – Software Developers Kit

6 Embedded BioAPI environment

6.1 Operating environment of Embedded BioAPI

Embedded BioAPI provides a standard interface for biometric modules that are provided to be used in an embedded system. Such systems are usually low profile, both in memory and computational power, which does not allow them to implement BioAPI (ISO/IEC 19784-1), or its Frameworkless version (see ISO/IEC 19784-1/Amd.2).

This means that the interface defined in this International Standard is a direct connection between an application and such biometric modules, without the use of operating system solutions and high level programming languages. The definition of this interface is given by specifying the services to be provided, as well as the message formats for the commands to be sent to the biometric modules, and the responses from them. Low level implementation details are not specified. See Annexes B and C for examples of such details.

To better understand the environment where Embedded BioAPI is to be applied, consider the example illustrated in Fig. 1. This figure represents the module block of a Point of Service (PoS) terminal (such as those used in shops for paying with a credit card), that integrates two biometric modalities in a monomodal way (e.g. fingerprint and handwritten signature). According to the application design, one of each modality will be chosen, depending on the context.

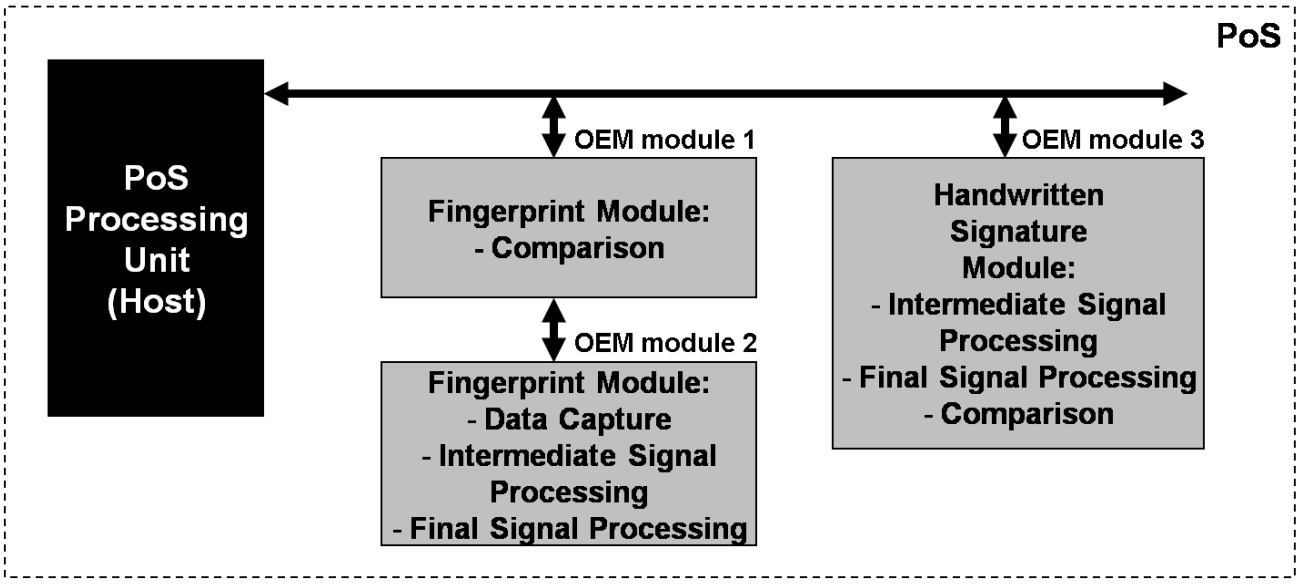

Figure 1 — Example of a Biometric System based on Embedded BioAPI subcomponents

In order to implement such a PoS terminal, the manufacturer must include all required biometric algorithms and data capture devices in the system. As the PoS is already an embedded system with limited processing capabilities, the microcontroller used may not be able to accommodate the complexity of such algorithms. Therefore, the PoS manufacturer chooses to acquire some hardware modules from biometric technology manufacturers, which include the required services. Consider the example illustrated in Fig. 1, where:

— The handwritten signature solution is provided as a monolithic hardware module which provides all required services, requiring only the supply (by a data capture device) of the raw biometric sample (following ISO/IEC 19794-7), and the user's enrolled template (by the PoS Processing Unit).

— The fingerprint solution is chosen to be the cascading of two monolithic modules, with the first one supporting all required services and comparison, but needing the connection to the second module for data capture device interfaces. The second module contains the data capture device and final signal processing algorithms (following ISO/IEC 19794-2). The first module does not need to be supplied with biometric samples, but it does need to be provided with the user's enrolled template in order to perform the comparison with the processed biometric sample extracted by the second module.

In such examples, modules have to provide all biometric services required, such as:

— Fingerprint:

  — Capture and Comparing against the user's template

— Handwritten Signature:

  — Comparison: providing the user's raw biometric sample and his/her stored template, which will be stored in a database or in a personal device such as a smart card.

Other systems and modules may need other functions, as shown in the services and commands Clause 9. For example, they may require template storage capability or enrolment.

It is also important to note that this International Standard only specifies API requirements to handle security but does not specify any security algorithms or key exchange mechanisms to be supported. Such security mechanisms are to be determined, for example, within communities of users or vendors. This International Standard recommends the use of standardized security mechanisms, such as the ones provided by ISO/IEC JTC1 SC27.

From a formal point of view, Embedded BioAPI is a set of interfaces derived from the full BioAPI (ISO/IEC 19784-1), but customized to embedded environments. It differs from BioAPI in the following ways:

a) It requires no framework component and does not utilize a component registry (i.e. there is a direct interface between the application and the Biometric Service Provider (BSP) components).

b) It includes a subset of the full BioAPI functionality (e.g. it does not support a one-to-many comparison capability or related primitives).

c) It minimizes the states to be handled.

d) Lack of callback mechanisms.

e) Data handles are not used.

f) It does not support a function provider interface (FPI).

g) Options are minimized.

It does, however, remain biometric-technology neutral and provides the basic functions required to perform biometric enrolment and verification through a generic Interface. Data formats (Biometric Data Information Records) shall be consistent with ISO/IEC 19784-1, and the relevant part (for those modalities that has a part published) of ISO/IEC 19794.

Embedded BioAPI conformant components generally consist of software or firmware associated with a specific hardware data capture device.

Embedded systems are typically characterized by constraints that may include:

a) Memory/storage limitations

b) Processor size/speed limitations

c) Limited operating system (OS) support

d) Single, hardwired data capture device

e) Standalone (unconnected to a network)

6.2 Security in Embedded BioAPI

This International Standard defines two kinds of devices:

— Type A: devices that do not implement any kind of security mechanism. This may be due to lack of processing capabilities, use for convenience rather than security purposes, etc..

— Type B: devices that implement security mechanisms for confidentiality, integrity, and/or ACBio generation. The supported mechanisms, the supported algorithm for each of the mechanisms, and the key information are dependent on the device, and shared with the application in advance of the execution. If encryption is supported, the BDB of the BIR shall be encrypted. If integrity is supported, it shall be applied to the concatenation of the SHB and the BDB.. If encryption and integrity are both supported, encryption shall be done first.

NOTE Use of Type B devices is highly recommended. For these devices, the following paragraph defines requirements for handling security, but the mechanisms to be used are out of the scope of this International Standard.

In Type B devices, biometric data shall be exchanged using the Security Block in the BIR, as defined in Clause 10. Therefore, biometric data shall be exchanged using the Security Block in the BIR, as defined in Clause 10. Security mechanisms in communication can be added at the low-level protocol, which is out of the scope of this International Standard. As Type A devices do not implement security mechanisms, all biometric information exchanged shall be used without a Security Block in the BIR, as defined in ISO/IEC 19784-1/Amd. 3 and ISO/IEC 19785-1.

7 Embedded BioAPI general architecture

The general architecture (where an Embedded BioAPI conformant biometric module is used) is composed of two elements, which need to interact in order to provide biometric-related functionality. One of these elements is a General Processing Unit, also called a host, that provides the top-level functionality of the whole device, and needs to connect to an Embedded BioAPI subcomponent (i.e. biometric module) that performs some or all biometric operations (processing, storage, enrolment and/or verification).

Embedded BioAPI is intended to support the integration of embedded biometric Embedded BioAPI subcomponents into a host device. Two options exist for this integration:

1) Monolithic Embedded BioAPI subcomponents (Type 1): A single Embedded BioAPI subcomponent includes the biometric data capture device, storage, and algorithms.

2) Compartmental Embedded BioAPI subcomponents (Type 2): A single Embedded BioAPI subcomponent which includes at least one but not all of the following capabilities:

   i) Biometric data capture device – ability to sense and capture the raw biometric data.

   ii) Biometric storage – onboard storage of template data.

   iii) Biometric algorithms – the ability to process and compare biometric data.

These two options are graphically represented in Figure 2.

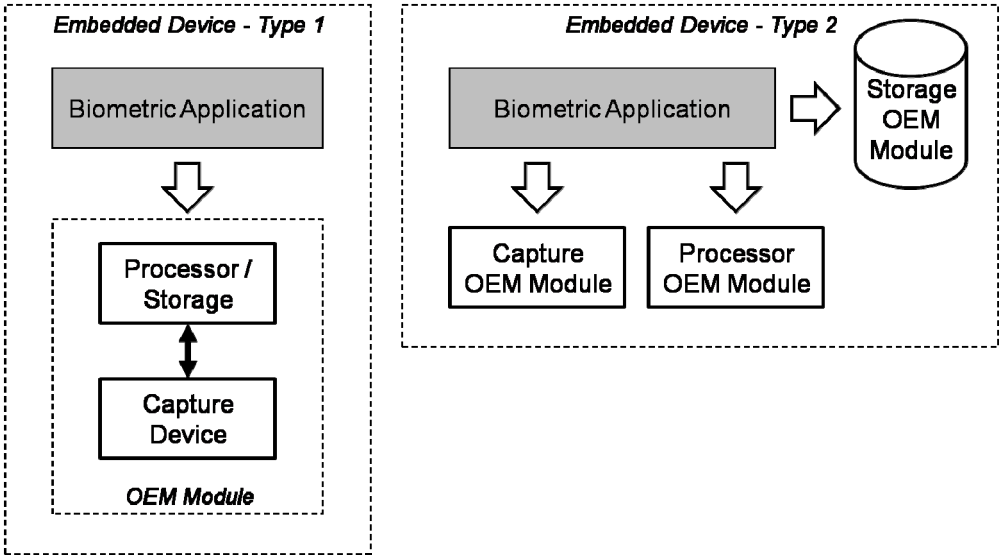

Figure 2 — Different ways to implement OEM biometric modules

An example of a Type 1 (monolithic) module is an OEM fingerprint data capture device/board designed to be installed into a remote control device and containing “full” biometric functionality including the fingerprint data capture device, processor, firmware (containing the biometric processing and comparison algorithms), and onboard storage for up to 5 fingerprint templates.

An example of a Type 2 (compartmentalized) implementation is an OEM commodity CCD camera capable of capturing facial photo images (i.e. the biometric data capture device) to be embedded into a home physical access device and a biometric module capable of processing and comparing facial images produced by that camera. In this case, the template storage could occur and be controlled separately by the application or be performed by the biometric module.

Both types of modules can be generalized in the following figure (Figure 3). As can be seen, a module conformant with Embedded BioAPI is a hardware module that performs some or all of the biometric functions required by a host system and that communicates with such host system through a standardized interface, herein specified as Embedded BioAPI. Optionally, that same module may have the possibility of contacting directly another Embedded BioAPI biometric module.

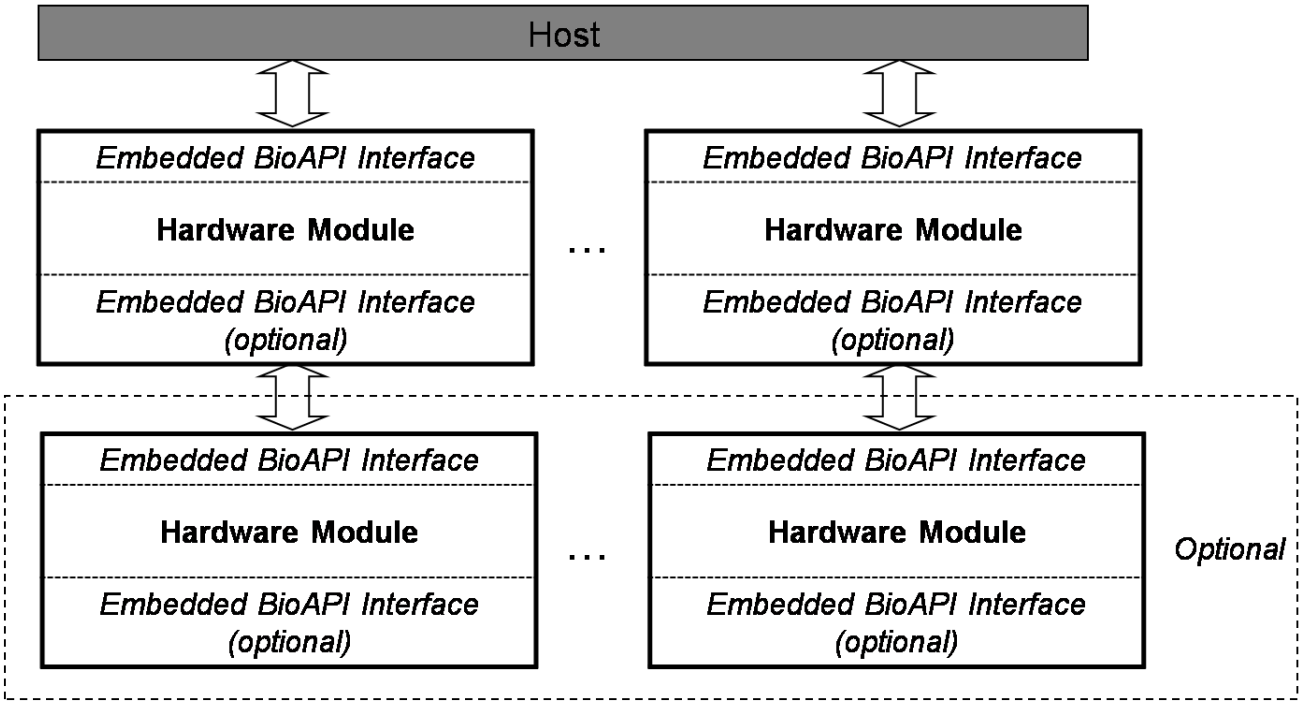

Figure 3 — General Architecture of an Embedded BioAPI system

If a BioAPI (ISO/IEC 19784-1) conformant system needs to use a device conformant with this international standard, the developer may treat the Embedded BioAPI device as a Unit within BioAPI, and develop the corresponding Biometric Service Provider (as defined in ISO/IEC 19784-1). Such a Biometric Service Provider will have to be conformant with ISO/IEC 19784-1 for the interface within the Application or the BioAPI Framework, and, at the same time, conformant with this international standard when interfacing directly with the Embedded BioAPI device. In other words, the developer will have to implement an ISO/IEC 19784-1 wrapper to the functions that connect directly with the Embedded BioAPI device.

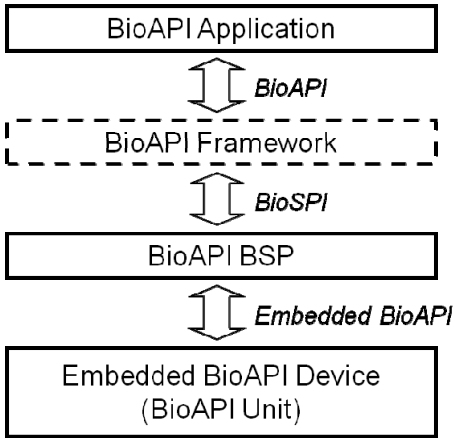

Figure 4 — Relationship between Embedded BioAPI and BioAPI components

8 Frames structure

When considering the limitations of biometric modules and/or hosts, major considerations regarding communication messages are:

— Data transmission between modules could be as large as 65KB for processed biometric samples and templates, and even larger for raw data (e.g. an infrared eye image for iris recognition).

  — To determine a fixed length is not viable. Therefore communication is split into frames, each of them of variable length, but with a length-field of 2 bytes.

  — A protocol for handling frames is needed, as well as defining the frame structure.

— Security mechanisms, such as ways to preserve confidentiality, integrity and authenticity of biometric data, shall be allowed by the defined interface.

Additionally there are considerations coming from Embedded BioAPI subcomponent requirements that affect the design of the communication messages, such as:

— Processing times vary significantly depending on the application. Timeouts should handle response times as low as 100 ms, and as large as seconds, especially if human action is involved.

In an Embedded BioAPI environment, a single-master/multiple-slave protocol is defined, where the master is the host device and the slaves are the different biometric modules. In case of a cascaded module configuration, the biometric module can connect to another module. Therefore, it shall implement the Embedded BioAPI Interface in master mode to communicate with such a module, as seen in Figure 4. For the definition of Embedded BioAPI interface, section 9 describes some of its fields.

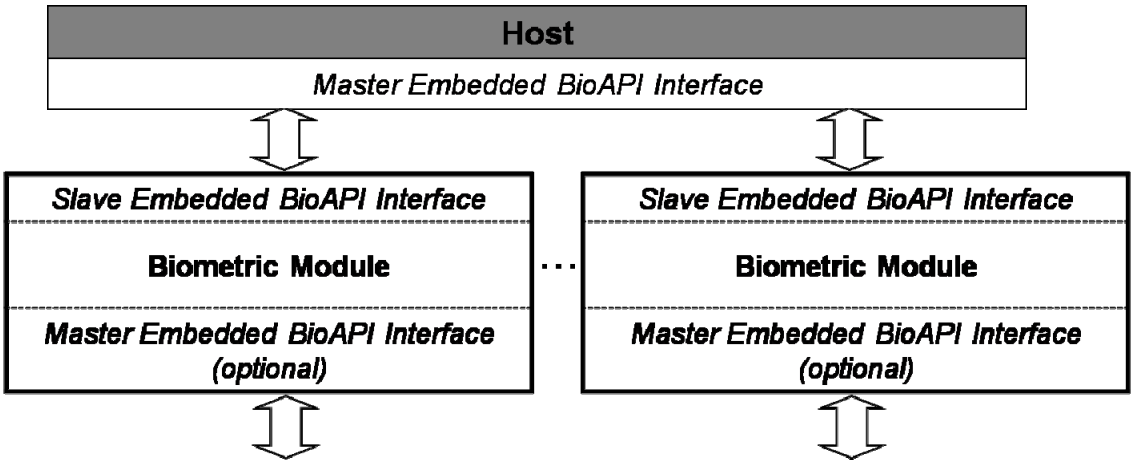

Figure 5 — Master / Slave configuration of the BioAPI Interface

As the number of modules to be considered in an application related to Embedded BioAPI is restricted, no real address is used, but an identification field is included (see Table 3 — Slave ID field [S] coding). Also, data is transferred at both the command level and also in responses. For an embedded system, it is expected that data is kept as small as possible, so that storage and transmission requirements are low enough. With that in mind, the amount of data to be transferred can be limited initially to 2^{16} bytes, so a data length field of 2 bytes is needed.

For a single-master / multiple-slaves system, the protocol is defined as a command/response pair, where commands are always sent by the host, and answers are always provided by the addressed slave. Commands, as shown in Clause 9, are less than 256, so a single byte field for the command identification is needed.

Data to be exchanged shall be included in the Data Block. When the information to be exchanged is biometric data, such data shall be coded as a BIR, as defined in ISO/IEC 19784-1/Amd.3, ISO/IEC 19785-1, and following ACBio procedures as specified in ISO/IEC 24761. As stated in Clause 6.1, for Type A devices, the BIR shall contain neither a Security Block nor an ACBio instance. In the case of Type B devices, the BIR shall have a Security Block as specified in Clause 6.

The command frame has the following structure:

Table 1 — Structure of the Command Frame

Slave ID [S] (1 byte)	Command [C] (1 byte)	Data Length [L] (2 bytes)	Data [D] (L bytes)

And the response frame is:

Table 2 — Structure of the Response Frame

Slave ID [S] (1 byte)	Status/Error [E] (1 byte)	Data Length [L] (2 bytes)	Data [D] (L bytes)

Error handling is in accordance with the rules of the low-level protocol used.

In order to concatenate frames as to exchange data longer than 2^{16} bytes, the [S] field has, as its most significant bit, a flag indicating with a '1' that further frames are to be sent, and with a '0', that no further frame is to be sent. In case of concatenation of frames, all further frames have to maintain intact their second byte (Command or Status/Error) related to the first of the frames in the concatenation, as well as the 7 least-significant bits of the [S] field.

9 Patron format for Embedded BioAPI

The patron format that shall be used within the scope of this International Standard is the 'Fixed-length-fields, bit-oriented patron format using presence bit-map' patron format, as defined in ISO/IEC 19785-3:2007, Clause 10.

10 Security block format for Embedded BioAPI

10.1 Security Block format owner

ISO/IEC JTC 1/SC 37

10.2 Security Block format owner identifier

257 (0101Hex). This has been allocated by the Registration Authority for ISO/IEC 19785-2.

10.3 Security Block format name

ISO/IEC JTC 1/SC 37 security block format for Embedded BioAPI.

10.4 Security Block format identifier

1 (0001 Hex). This has been registered in accordance with ISO/IEC 19785-2 when BER encodings (See ISO/IEC 8825-1) are applied.

m (000m Hex). This has been registered in accordance with ISO/IEC 19785-2 when canonical XER encodings (See ISO/IEC 8825-3) are applied.

10.5 ASN.1 object identifier for this security Block format

10.5.1 The case of BER encodings

```
{iso registration-authority embedded-bioapi(29164) organizations(0) jtc-sc37 (257) sb-format(3) ber-encoding(1)}
```

or, in XML value notation,

```
1.1.29164.0.257.3.1
```

10.5.2 The case of canonical XER encodings

```
{iso registration-authority embedded-bioapi(29164) organizations(0) jtc-sc37 (257) patron-format(3) xer-encoding(2)}
```

or, in XML value notation,

```
1.1.29164.0.257.3.2
```

10.6 Domain of use

This clause provides a definition of the security block format that is to be used for Embedded BioAPI.

10.7 Version identifier

This patron format specification has a version identifier of (major 0, minor 0).

10.8 CBEFF version

This specification conforms to CBEFF version (major 2, minor 0).

10.9 General

In **SecurityBlockForEmbeddeBioAPI**, there are two fields, **subBlockForIntegrity** and **acBioInstance**. Since security algorithms and/or keys used in security operations are to be exchanged in advance, there is no need for them to be included in the SB. If encryption is supported on the device, the BDB is encrypted. If integrity is supported on the device, the digital signature or authentication code for the concatenation of the SBH and BDB is set into the field **subBlockForIntegrity**. If ACBio generation is supported, the ACBio instance is set into the field **acBioInstance**.

10.10 Specification

```
SECURITY-FORMAT-FOR-EMBEDDED-BIOAPI
       {iso standard embeddedbioapi(29164)modules(0) sbformat(1) rev(0)}

DEFINITIONS AUTOMATIC TAGS ::= BEGIN
IMPORTS

-- ISO/IEC 24761 Authentication context for biometrics
       ACBioInstance
       FROM AuthenticationContextForBiometrics {
           iso(1) standard(0) acbio(24761) module(1) acbio(2) rev(0)} ;
```

```
SecurityBlockForEmbeddeBioAPI ::= SEQUENCE {
        subBlockForIntegrity OCTET STRING    OPTIONAL,
        acBioInstance ACBioInstance     OPTIONAL
}

END
```

11 Data types, formats and coding

11.1 Slave ID field [S]

The [S] field shall be coded according to the following table:

Table 3 — Slave ID field [S] coding

b7	b6	b5	b4	b3	b2	b1	b0	
0								No further frame to be sent
1								Further frame is to be sent
	x	x	x	x	x			Modality in 5 bits (following coding in 11.4)
						0	0	Capture functionality module
						0	1	Signal processing functionality
						1	0	Comparison functionality
						1	1	Multiple functionalities

NOTE: As in those environments where Embedded BioAPI is to be applied, no more than one module for modality and functionality is expected, no real addressing is needed, but just a way to identify among the different modules installed.

11.2 Command field [C]

Regarding the functions defined in this International Standard, their coding is done hierarchically as shown in the following table:

Table 4 — Command field [C] coding

b7	b6	b5	b4	b3	b2	b1	b0	Hex value	
0	0								**Management Commands**
0	0	0	1	1	0	0	1	19	emBioAPI_BSPAttach
0	0	0	0	0	1	1	0	06	emBioAPI_BSPDetach
0	0	0	0	0	0	0	1	01	emBioAPI_QueryUnit
0	0	0	0	0	0	1	0	02	emBioAPI_ControlUnit
0	0	1	0	1	0	1	0	2A	emBioAPI_Cancel
0	0	x	x	x	x	x	x	xx	RFU
0	1								**Template Management Commands**
0	1	0	0	0	0	0	1	41	emBioAPI_DBStoreBIR
0	1	0	0	0	0	1	0	42	emBioAPI_DBGetBIR
0	1	0	0	0	1	0	0	44	emBioAPI_DeleteBIR
0	1	x	x	x	x	x	x	xx	RFU
1	0								**Enrolment Commands**
1	0	0	0	0	0	0	1	81	emBioAPI_Capture4Enrol
1	0	0	0	0	0	1	0	82	emBioAPI_GetBIR
1	0	0	0	1	0	0	0	88	emBioAPI_CreateTemplate
1	0	x	x	x	x	x	x	xx	RFU

1	1								**Biometric Process Commands**
1	1	0	0	0	0	0	1	C1	emBioAPI_Capture
1	1	0	0	0	0	1	0	C2	emBioAPI_Process
1	1	0	0	0	0	1	1	C3	emBioAPI_CaptureProcess
1	1	0	0	0	1	0	0	C4	emBioAPI_VerifyExternal
1	1	0	0	0	1	1	0	C6	emBioAPI_VerifyMatchExternal
1	1	0	0	0	1	1	1	C7	emBioAPI_ProcessVerifyMatchExternal
1	1	0	0	1	1	0	0	CC	emBioAPI_VerifyInternal
1	1	0	0	1	1	1	0	CE	emBioAPI_VerifyMatchInternal
1	1	0	0	1	1	1	1	CF	emBioAPI_ProcessVerifyMatchInternal
1	1	x	x	x	x	x	x	xx	RFU

11.3 Status/Error field [E]

Coding of the Status/Error field shall be done according to the following table

Table 5 — Status/Error field [E] coding

b7	b6	b5	b4	b3	b2	b1	b0	C Definition	
0									**Command processed correctly**
0	0	0	0	0	0	0	0	NO_ERROR	No error
0	0	0	0	0	0	1	0	MATCH	Match
0	1	1	1	1	1	0	1	NO_MATCH	No Match
0	0	0	0	0	1	0	1	FTA	Failed to Acquire (FTA)
0	0	0	0	1	0	0	1	FTE	Failed to Enrol (FTE)
0	0	0	1	0	0	0	1	NO_QUALITY	Quality requirements not accomplished
0	0	1	0	0	0	1	1	WAIT	Wait, still processing
0	x	x	x	x	x	x	x		RFU
1									**Error in command processing**
1	0	0	1	1	0	0	1	TIME_OUT	Time-out
1	0	0	1	1	0	1	0	UNK_COMMAND	Command not supported
1	0	0	1	1	1	0	0	INTEGR_FAIL	Integrity check failed
1	0	0	1	1	1	0	1	SECURITY_FAIL	Security check failed
1	0	0	1	1	1	0	0	UNDEFINED	Unexpected/undefined
1	x	x	x	x	x	x	x		RFU

11.4 Biometric modalities coding

Coding of biometric modalities is done in accordance with the BioAPI_BIR_BIOMETRIC_TYPE defnition in subclause 7.58 of ISO/IEC 19784-1:2006/AMD1. In order to truncate the 32 bit value into 5 bits, the following rule applies:

1) The modality covered shall be included in section 7.58 ISO/IEC 19784-1:2006/AMD1, as a 32 bit number in the form 2^x.

2) [S] shall be the exponent x incremented in one unit ([S] = x+1), and coded as the 5 least significant bits.

3) [S] = 0 is Reserved for Future Use (RFU)

12 Commands definition

This clause covers the commands included in Embedded BioAPI. The definition of all the commands is presented in the following way:

- Description of the command;
- Input parameters to be sent within the command;
- Response sent by the biometric module;
- Frame structure of the frames involved (both command and response). In this International Standard the following notation rules apply:
  - Hexadecimal values are represented with numbers and capitalized letters;
  - Variable symbols are represented with non-capitalized letters;
  - If during command processing, an error occurs, the response frame does not carry any data.
- Header of the C function for SDK development.
  - The returned value is, in all cases, the Status/Error code.
  - For each of the parameters, a comment telling whether it is an input (IN) or an output (OUT) parameter. Input-output (INOUT) fields are not used in this international standard.

Commands have been grouped into the following categories:

- Module Management
- Template Management
- Enrolment
- Biometric Processing

All biometric data exchanged by the commands shall be in accordance with ISO/IEC 19785-1 and the relevant part of ISO/IEC 19794 series of Standards (for those modalities where a part in ISO/IEC 19794 has been published). Where a compact format (sometimes referred as card format) is available, its use is recommended.

12.1 Management commands

12.1.1 emBioAPI_BSPAttach (mandatory)

Description: Tells the module to initialize itself, opening the offered services and initializing all security functions for encryption, integrity, and ACBio instance generation. This command is to be called any time a session is started (power on, session change, etc). Without being called, the rest of the commands shall not work, with the exception of QueryUnit.

The same function is used for Type A and Type B devices. As Type A devices do not support security mechanisms, they shall discard the security-related fields sent by the host, and shall fill with NULL characters the security-related fields in the response frame. In order to detect that the module connected is of Type A, the host shall check that both the Random Number (RN_B) and the auxiliary data sent by the device are filled with 0x00 characters.

Parameters:

- Key set ID: 1 byte indicating the number of the keyset to be used. A value of 0x00 shall be used for those devices not implementing several sets of keys and for those that have a default set defined. This field shall be ignored by all Type A devices, although the host shall send the field in all cases.
- Algorithm ID: 1 byte indicating the number of the cryptographic algorithm to be used. A value of 0x00 shall be used for those devices not implementing several algorithms and for those that have a default algorithm defined. Type A devices shall code this field as 0x00. This field shall be ignored by all Type A devices, although the host shall send the field in all cases.
- Random Number (RN_A), to be used internally by the module for session key calculation (16 bytes). This field shall be ignored by all Type A devices, although the host shall send the field in all cases.
- BioAPI_ACBio_PARAMETERS (as defined in ISO/IEC 19784-1/Amd.3)

Response: Initialization information, including information generated for session key computation for secure data exchange. Also the information about biometric modality used, as well as services offered.

- Status code or error (see Table 5 — Status/Error field [E] coding)
- Random Number (RN_B), generated by the biometric module for session key calculation (16 bytes). For Type A devices, this field shall be filled with NULL bytes.
- Length of auxiliary data
- Auxiliary data, e.g. the result of ciphering the RN_A with the session key calculated. For Type A devices, this field shall be filled with NULL bytes.

Messages Frames:

Command (hex): `aa 19 vv vv kk gg rr ... rr hh hh jj ... jj`

Response (hex): `aa ee 00 20 bb ... bb ll ll cc ... cc`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code
vv vv – data length in command (depends on length of ACBio_Parameters)
kk – Key Set ID
gg – Algorithm ID
rr – Random Number RN_A (16 bytes)
hh hh – Length of ACBio Parameters (2 bytes)
jj ... jj – ACBio_Parameters (as per ISO/IEC 19784-1/Amd.3)
bb – Random Number RN_B (16 bytes)
ll ll – Length of auxiliary data (2 bytes)
cc – Auxiliary data (ll ll bytes)

C Function Header:

```
unsigned char emBioAPI_BSPAttach (
        unsigned char    slaveID,                    // IN
        unsigned char    keysetID,                   // IN
        unsigned char    algid,                      // IN
        unsigned char    *RNA,                       // IN
        unsigned short   length_acbio_parameters,    // IN
        unsigned char    *acbio_parameters,          // IN
        unsigned char    *RNB,                       // OUT
        unsigned short   *length_aux_data            // OUT
        unsigned char    *auxiliarydata              // OUT
        );
```

12.1.2 emBioAPI_BSPDetach (mandatory)

Description: Tells the module to shut down. When this function is called in a Type B biometric module, all session keys shall be discarded, and the security status lowered to the minimum level.

Parameters: None

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding)

Messages Frames:

Command (hex): `aa 06 00 00`

Response (hex): `aa ee 00 00`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code

C Function Header:

```
unsigned char emBioAPI_BSPDetach (
        unsigned char slaveID
        );
```

12.1.3 emBioAPI_QueryUnit (mandatory)

Description: Gets the properties of the module. It provides information on capabilities, configuration, and state. It provides its biometric modality, the number of templates that can be stored inside the module, the number of templates currently stored, and all supported functionalities.

emBioAPI_QueryUnit can be called before or after emBioAPI_BSPAttach.

Parameters: None

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding)
- Biometric modality covered (see Biometric modalities coding in section 11.4)
- Template storage capacity (in 2 bytes)
- Number of templates currently stored
- Data format implemented
- Number of functions, which is the length of the “Functions supported” parameter
- Functions supported by the module, i.e. a sequence of bytes enumerating all functions accepted by the biometric module (see Table 4 — Command field [C] coding)

Messages Frames:

Command (hex): `aa 01 00 00`

Response (hex): `aa ee ww ww mm cc cc tt tt dd dd nn ff ... ff`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code
ww ww – data length in response (depends on number of functions supported)
mm – Modality (1 byte)
cc cc –Template storage capacity (2 bytes)
tt tt – Number of templates stored (2 bytes)
dd dd – Data format supported coded as defined in ISO/IEC 19784-1
nn – Number of functions returned (1 byte)
ff ... ff – Functions supported

C Function Header:

```
unsigned char emBioAPI_QueryUnit(
        unsigned char    slaveID,            // IN
        unsigned char    *modality,          // OUT
        unsigned short   *capacity,          // OUT
        unsigned short   *num_templates,     // OUT
        unsigned short   *data_format,       // OUT
        unsigned char    *number_functions,  // OUT
        unsigned char    *functions          // OUT
        );
```

12.1.4 emBioAPI_ControlUnit

Description: Updates parameters in biometric module. One such parameter can be the comparison threshold. For that reason this function is recommended to be used only in Type B devices with all security mechanisms available.

Parameters: — Parameters (vendor specific)

Response: — Status code or error (see Table 5 — Status/Error field [E] coding)

Messages Frames:

Command (hex): `aa 02 vv vv pp ... pp`

Response (hex): `aa ee 00 00`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code
vv vv – data length of parameters
pp ... pp – Parameters (vendor specific)

C Function Header:

```
unsigned char emBioAPI_ControlUnit(
        unsigned char    slaveID,            // IN
        unsigned short   parameters_length,  // IN
        unsigned char    *parameters         // IN
        );
```

12.1.5 emBioAPI_Cancel

Description: Cancel asynchronously the current running process. This command can be used while any time-consuming command is being executed (e.g. emBioAPI_Capture4Enrol, emBioAPI_Capture, emBioAPI_Process, etc.)

Parameters: None

Response:
— Status code or error (see Table 5 — Status/Error field [E] coding)

Messages Frames: Command (hex): `aa 2A 00 00`

Response (hex): `aa ee 00 00`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code

C Function Header:

```
unsigned char emBioAPI_Cancel(
        unsigned char    slaveID       // IN
        );
```

12.2 Template management commands

12.2.1 emBioAPI_DBStoreBIR (onboard template storage)

Description: Stores the input template in the internal biometric module database.

For Type B devices, the supported security functions are dependent on the devices. See 6.2 for details. If ACBio generation is supported, this command shall generate an ACBio instance.

Parameters:
— Length of template BIR
— Template BIR

Response:
— Status code or error (see Table 5 — Status/Error field [E] coding)
— Template number (2 bytes)

Messages Frames: Command (hex): `aa 41 vv vv tt ... tt`

Response (hex): `aa ee 00 02 nn nn`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
vv vv – Length of template BIR
tt ... tt – Template BIR
nn nn – Template number assigned

C Function Header:

```
unsigned char emBioAPI_DBStoreBIR (
        unsigned char    slaveID,                // IN
        unsigned short   template_BIR_length,    // IN
        unsigned char    *template_BIR,          // IN
        unsigned short   *template_number        // OUT
        );
```

12.2.2 emBioAPI_DBGetBIR (onboard template storage)

Description: Obtains the indicated (by number) template from the biometric module.

For Type B devices, the supported security functions are dependent on the devices. See 6.2 for details. If ACBio generation is supported, this command shall generate an ACBio instance.

Parameters:
- Template number (2 bytes)

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding)
- Length of template BIR
- Template BIR

Messages Frames: Command (hex): `aa 42 00 02 nn nn`

Response (hex): `aa ee vv vv tt ... tt`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
nn nn – Template number
vv vv – Length of template BIR
tt ... tt – Template BIR

C Function Header:

```
unsigned char emBioAPI_DBGetBIR (
        unsigned char    slaveID,                // IN
        unsigned short   template_number,        // IN
        unsigned short   *template_BIR_length,   // OUT
        unsigned char    *template_BIR           // OUT
        );
```

12.2.3 emBioAPI_DeleteBIR (mandatory for devices with template storage capabilities)

Description: Erases all enrolled templates or the indicated (by number) template.

This command does not exchange biometric information, as all data is handled internally. Therefore no security mechanisms are requested for Type B devices.

Parameters:
- Number of template to be erased (2 bytes)
  - If all templates are to be erased, use 0xFFFF

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding)

Messages Frames: Command (hex): `aa 96 00 02 nn nn`

Response (hex): `aa ee 00 00`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code
nn – Number of template to be erased

C Function Header:

```
unsigned char emBioAPI_DeleteBIR (
        unsigned char    slaveID,            // IN
        unsigned short   template_number     // IN
        );
```

12.3 Enrolment commands

12.3.1 emBioAPI_Capture4Enrol

Description: Performs a biometric capture (using onboard data capture device), keeping the information internal to the module for later enrolment processing. The number of times this function is called depends on the number of samples the module needs to perform enrolment.

As this operation involves user interaction, the biometric module manufacturer shall consider time-out values in order to cancel operation, reporting that situation in the Status code returned.

This command does not exchange biometric information, as all data is handled internally. Even for Type B devices, no security functions are executed including ACBio generation.

Parameters: None

Response: — Status code or error (see Table 5 — Status/Error field [E] coding)

Messages Frames: Command (hex): `aa 81 00 00`

Response (hex): `aa ee 00 00`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code

C Function Header:

```
unsigned char emBioAPI_Capture4Enrol(
        unsigned char slaveID      // IN
        );
```

12.3.2 emBioAPI_GetBIR

Description: Sends a sample BIR to the module to keep the information for a later enrolment process. The number of times that this function is called depends on the number of samples the module needs to perform enrolment. Depending on module capabilities, input data can be a raw sample, an intermediate signal, or its corresponding processed biometric sample.

For Type B devices, the supported security functions are dependent on the devices. For Type B devices which support ACBio generation, an ACBio instance shall be generated after the template is created.

Parameters:
— Length of sample BIR
— Sample BIR

Response: — Status code or error (see Table 5 — Status/Error field [E] coding).

Messages Frames: Command (hex): `aa 82 vv vv ss ... ss`

Response (hex): `aa ee 00 00`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code
vv vv – Sample BIR length
ss ... ss –Sample BIR

C Function Header:

```
unsigned char emBioAPI_GetBIR (
        unsigned char    slaveID,            // IN
        unsigned short   sample_BIR_length,  // IN
        unsigned char    *sample_BIR         // IN
        );
```

12.3.3 emBioAPI_CreateTemplate

Description: Performs an enrolment to create a template and stores the template within the module. To execute this function, either emBioAPI_Capture4Enrol or emBioAPI_GetBIR functions have to be called in advance. The input samples shall be processed and the processed samples shall be stored in the module. The return value is the internally assigned number of the created template.

This command does not exchange biometric information, as all data is handled internally. For Type B devices which support ACBio generation, an ACBio instance should be generated after the template is created.

Parameters: None

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding)
- Template number (if success)

Messages Frames: Command (hex): `aa 88 00 00`

Response (hex): `aa ee 00 ww nn nn`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code
ww = 02 if success, 00 if error
nn nn – Template number assigned (only if success)

C Function Header:

```
unsigned char emBioAPI_CreateTemplate (
        unsigned char    slaveID,            // IN
        unsigned short   *template_number    // OUT
        );
```

12.4 Biometric process commands

12.4.1 emBioAPI_Capture

Description: Performs a biometric capture (using onboard data capture device), returning the sample BIR.

As this operation involves user interaction, the biometric module manufacturer shall consider time-out values in order to cancel operation, reporting that situation in the Status code returned.

For Type B devices, the supported security functions are dependent on the devices. See 6.2 for details. If ACBio generation is supported, this command shall generate an ACBio instance.

Parameters: None

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding)
- Length of sample BIR
- Sample BIR. Depending on module capabilities, this outcome can be a raw sample or intermediate signal processed data, but it shall not be a processed biometric sample.

Messages Frames: Command (hex): `aa C1 00 00`

Response (hex): `aa ee ww ww ss ... ss`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code
ww ww – Sample BIR length
ss ... ss – Sample BIR

C Function Header:

```
unsigned char emBioAPI_Capture (
        unsigned char    slaveID,             // IN
        unsigned short   *sample_BIR_length, // OUT
        unsigned char    *sample_BIR          // OUT
        );
```

12.4.2 emBioAPI_Process

Description: Processes a sample BIR to create comparable recognition data (processed sample BIR). Depending on module capabilities, the input sample can be a raw sample or an intermediate signal processed one.

For Type B devices, the supported security functions are dependent on the devices. See 6.2 for details. If ACBio generation is supported, this command shall generate an ACBio instance.

Parameters:
- Length of sample BIR data
- Sample BIR data

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding)
- Length of processed sample BIR
- Processed sample BIR. This data shall be ready to enter a comparison module.

Messages Frames:

Command (hex): `aa C2 vv vv ss ... ss`

Response (hex): `aa ee ww ww ff ... ff`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
vv vv – Sample BIR length
ss ... ss – Sample BIR data
ww ww – Processed sample BIR length
ff ... ff – Processed sample BIR data

C Function Header:

```
unsigned char emBioAPI_Process (
        unsigned char    slaveID,                   // IN
        unsigned short   sample_BIR_length,         // IN
        unsigned char    *sample_BIR,               // IN
        unsigned short   *proc_sample_BIR_length,   // OUT
        unsigned char    *proc_sample_BIR           // OUT
        );
```

12.4.3 emBioAPI_CaptureProcess

Description: Performs a biometric capture (using onboard data capture device), returning its processed sample BIR.

As this operation involves user interaction, the biometric module manufacturer shall consider time-out values in order to cancel operation, reporting that situation in the Status code returned.

For Type B devices, the supported security functions are dependent on the devices. See 6.2 for details. If ACBio generation is supported, this command shall generate an ACBio instance.

Parameters: None

Response:

— Status code or error (see Table 5 — Status/Error field [E] coding)
— Length of processed sample BIR
— Processed sample BIR. This data shall be ready to enter a comparison module.

Messages Frames:

Command (hex): `aa C3 00 00`

Response (hex): `aa ee ww ww ff ... ff`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code
ww ww – Processed sample BIR length
ff ... ff – Processed sample BIR data

C Function Header:

```
unsigned char emBioAPI_CaptureProcess (
        unsigned char    slaveID,                   // IN
        unsigned short   *proc_sample_BIR_length,   // OUT
        unsigned char    *proc_sample_BIR           // OUT
        );
```

12.4.4 emBioAPI_VerifyExternal

Description: Performs a biometric capture (using onboard data capture device), processes the sample BIR, and compares it with the template sent from outside of the Embedded BioAPI subcomponent.

As this operation involves user interaction, the biometric module manufacturer shall consider time-out values in order to cancel operation, reporting that situation in the Status code returned.

For Type B devices, the supported security functions are dependent on the devices. See 6.2 for details. If ACBio generation is supported, this command shall generate an ACBio instance.

Parameters:
- Length of template BIR
- Template BIR used for comparison.

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding), showing comparison binary decision in case of no other error.

Messages Frames: Command (hex): `aa C7 vv vv tt ... tt`

Response (hex): `aa ee 00 00`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code
vv vv – Template BIR length
tt ... tt – Template BIR

C Function Header:

```
unsigned char emBioAPI_Verify (
        unsigned char    slaveID,                // IN
        unsigned short   template_BIR_length,    // IN
        unsigned char    *template_BIR           // IN
        );
```

12.4.5 emBioAPI_VerifyMatchExternal

Description: Compares a processed sample BIR with the template sent by the external world.

For Type B devices, the supported security functions are dependent on the devices. See 6.2 for details. If ACBio generation is supported, this command shall generate an ACBio instance.

Parameters:
- Length of processed sample BIR data
- Processed sample BIR
- Length of template BIR
- Template BIR used for comparison

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding), showing comparison binary decision in case of no other error.

Messages Frames: Command (hex): `aa C4 vv vv hh hh ss ... ss jj jj tt ... tt`

Response (hex): `aa ee 00 00`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
vv vv – Data frame length (= 2 + hhhh 2 + jjjj)
hh hh – Processed sample BIR length
ss ... ss – Processed sample BIR data
jj jj – Template BIR length
tt ... tt – Template BIR data

C Function Header:

```
unsigned char emBioAPI_VerifyMatchExternal (
        unsigned char    slaveID,                    // IN
        unsigned short   feature_vector_BIR_length, // IN
        unsigned char    *feature_vector_BIR,        // IN
        unsigned short   template_BIR_length,       // IN
        unsigned char    *template_BIR               // IN
        );
```

12.4.6 emBioAPI_ProcessVerifyMatchExternal

Description: Processes a sample BIR and compares it with the template, both being sent by the host to the embedded device.

For Type B devices, the supported security functions are dependent on the devices. See 6.2 for details. If ACBio generation is supported, this command shall generate an ACBio instance.

Parameters:
- Length of sample BIR data
- Sample BIR data
- Length of template BIR
- Template BIR used for comparison

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding), showing comparison binary decision in case of no other error.

Messages Frames: Command (hex): `aa C6 vv vv hh hh ss ... ss jj jj tt ... tt`

Response (hex): `aa ee 00 00`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
vv vv – Data frame length (= 2 + hhhh + 2 + jjjj)
hh hh – Sample BIR length
ss ... ss – Sample BIR data
jj jj – Template BIR length
tt ... tt – Template BIR

C Function Header:

```
unsigned char emBioAPI_ProcessVerifyMatchExternal (
        unsigned char    slaveID,                 // IN
        unsigned short   sample_BIR_length,      // IN
        unsigned char    *sample_BIR,            // IN
        unsigned short   template_BIR_length,    // IN
        unsigned char    *template_BIR           // IN
        );
```

12.4.7 emBioAPI_VerifyInternal

Description: Performs a biometric capture (using onboard data capture device), processes the sample BIR, and compares it with templates stored in the module. If the input parameter is 0xFF, comparison shall be done with all templates stored. Otherwise, comparison is done only with the template whose internal assigned number is given as the input parameter.

As this operation involves user interaction, the biometric module manufacturer shall consider time-out values in order to cancel operation, reporting that situation in the Status code returned.

For Type B devices which support ACBio generation, an ACBio instance shall be generated after the template is created.

Parameters:
- Template number used for comparison (0xFFFF means all of them).

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding), showing comparison binary decision in case of no other error.
- List of template numbers matching the sample acquired (only in case Status is Match_OK, and input parameter was 0xFFFF).
- ACBio instance.

Messages Frames:

Command (hex): `aa CF 00 02 nn nn`

Response (hex): `aa ee vv vv mm ... mm uu uu ll ... ll`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
ee – Status/Error code
nn nn – Template number used for comparison (0xFFFF = all stored ones)
vv vv – Template numbers list length
mm ... mm – Template numbers list
uu uu – ACBio instance length
ll ... ll – ACBio instance

C Function Header:

```
unsigned char emBioAPI_VerifyInternal (
        unsigned char    slaveID,                          // IN
        unsigned short   biometric_reference_number,       // IN
        unsigned short   biometric_reference_list_length,  // OUT
        unsigned char    *biometric_reference_list,        // OUT
        unsigned short   output_BIR_length,                // OUT
        unsigned char    *output_BIR                       // OUT
        );
```

12.4.8 emBioAPI_VerifyMatchInternal

Description: Compares a processed sample BIR with templates stored in the module. If the input parameter is 0xFFFF, comparison shall be done with all templates stored. Otherwise, comparison is done only with the template whose internal assigned number is given as the input parameter.

For Type B devices, the supported security functions are dependent on the devices. See 6.2 for details. If ACBio generation is supported, this command shall generate an ACBio instance.

Parameters:
- Template number used for comparison (0xFFFF means all of them).
- Length of processed sample BIR
- Processed sample BIR

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding), showing comparison binary decision in case of no other error.
- Length of the list of templates matching
- List of templates matching

Messages Frames: Command (hex): `aa CE vv vv nn nn ss ... ss`

Response (hex): `aa ee rr rr mm ... mm`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
vv vv – Data frame length (= 2 + length of processed sample BIR)
nn nn – Template number used for comparison (0xFFFF = all stored ones)
ss ... ss – Processed sample BIR data
rr rr – Template numbers list length (in number of bytes)
mm ... mm – Template numbers list (each is 2 bytes)

C Function Header:

```
unsigned char emBioAPI_VerifyMatchInternal(
        unsigned char   slaveID,                    // IN
        unsigned short  template_number,            // IN
        unsigned short  feature_vector_length,      // IN
        unsigned char   *feature_vector_data,       // IN
        unsigned short  *template_list_length,      // OUT
        unsigned short  *template_list              // OUT
        );
```

12.4.9 emBioAPI_ProcessVerifyMatchInternal

Description: Processes a sample BIR and compares it with templates stored in the module. If the input parameter is 0xFFFF, comparison shall be done with all templates stored. Otherwise, comparison is done only with the template whose internal assigned number is given as the input parameter.

For Type B devices, the supported security functions are dependent on the devices. See 6.2 for details. If ACBio generation is supported, this command shall generate an ACBio instance.

Parameters:
- Template number used for comparison (0xFFFF means all of them).
- Length of sample BIR data
- Sample BIR data

Response:
- Status code or error (see Table 5 — Status/Error field [E] coding), showing comparison binary decision in case of no other error.
- List of template numbers that are verified against the sample BIR

Messages Frames:

Command (hex): `aa CE vv vv nn nn ss ... ss`

Response (hex): `aa ee ww ww mm ... mm`

where: aa – Slave ID (see Table 3 — Slave ID field [S] coding)
vv vv – Data frame length (= 2 + Sample BIR data length)
nn nn – Template number used for comparison (0xFFFF = all stored ones)
ss ... ss – Sample BIR data
ww ww – Template numbers list length (in number of bytes)
mm ... mm – Template numbers list (each one is 2 bytes)

C Function Header:

```
unsigned char emBioAPI_ProcessVerifyMatchInternal (
        unsigned char   slaveID,               // IN
        unsigned short  template_number,       // IN
        unsigned short  sample_BIR_length,     // IN
        unsigned char   *sample_BIR,           // IN
        unsigned short  *template_list_length, // OUT
        unsigned short  *template_list         // OUT
        );
```

Annex A
(normative)

Conformance Requirements

This Annex lists all the mandatory and conditional requirements specified in this International Standard, and serves as the basis for Conformance Testing. A.1 covers general statements, and subsequent sub-clauses cover the mandatory statements for each relevant condition.

A.1 General conformance statements

For all devices implementing Embedded BioAPI, they conform to this International Standard if they cover the following items as well as the relevant ones in the following subclauses:

- Data formats used for any biometric information shall be conformant to the relevant part of ISO/IEC 19794 series of standards, for those modalities where a part of ISO/IEC 19794 has been published.
- BIR formats shall be consistent with ISO/IEC 19784-1.
- All Embedded BioAPI implementations shall be conformant to those relevant standards dealing with security of biometric data and biometric devices.
    - If no detailed information is found, then an Embedded BioAPI implementation is conformant to this International Standard if it follows the security rules in 6.2.
- To be conformant with this International Standard, all Embedded BioAPI implementations shall implement a command/response frame as defined in Clause 8.
- All commands in any Embedded BioAPI implementation shall not be executed unless a successful emBioAPI_BSPAttach command has been successfully accomplished, with the exception of QueryUnit.
- The execution of emBioAPI_BSPDetach shall erase all session keys and shall take the current security level of the biometric module to a minimum, requiring overcoming security mechanisms in order to continue working with the device.
- All Embedded BioAPI commands implemented by any implementation of this International Standard shall be conformant to the relevant definitions in Clause 12.
- All SDK developed for Embedded BioAPI conformant biometric modules shall define all the covered functions as specified in Clause 12.
- All biometric data (samples and templates) shall be transmitted in an encrypted form for Type B devices.
- Slave ID field [S], Command field [C], Status/Error field [E] and Biometric Modality coding shall be done in accordance with Clause 11.
- All implementations of Embedded BioAPI shall include the following commands:
    - emBioAPI_BSPAttach
    - emBioAPI_BSPDetach
    - emBioAPI_QueryUnit

A.2 Conformance for modules with template storage capabilities

All implementations of Embedded BioAPI modules with the capability of storing templates, shall include the following functions:

- emBioAPI_DeleteBIR
- emBioAPI_DBStoreBIR

A.3 Conformance with security requirements

For Type B devices, security mechanisms shall be provided in order to achieve confidentiality, integrity and authenticity. For these devices the following conformance tests shall be performed:

- After detecting any failure in being conformant with any of the security mechanisms included in the biometric module, all session keys shall be erased and the module security status shall be taken to the minimum level.
- When the information to be exchanged is biometric data, such data shall be coded as a BIR, as defined in this International Standard.
- BIRs shall include a Security Block and other security information stated in Clause 10 of this International Standard.

For Type A devices, the BIR shall not contain a Security Block, neither an ACBio instance.

Annex B
(informative)

Examples of frame implementations

B.1 Introduction

This annex describes how to implement Embedded BioAPI using different communication interfaces.

B.2 Asynchronous serial communication

When using an UART (Universal Asynchronous Receiver Transmiter) Embedded BioAPI can be easily implemented by sending the proposed commands and responses in Clause 9. No further framing or enveloping are needed, unless further security mechanisms and/or integrity checks are requested. Communication should be implemented as a single master, multiple slave, where the modules are the slaves.

Apart from the command and response codings, the specific implementation shall define values for timeout between characters and timeout between any command and its response.

B.3 USB communication

Within USB communication, commands and responses have to be coded inside the data packets of the standard USB connection. This specification plus the recommendations for the Asynchronous serial communication are the only requirements for this implementation.

B.4 Two wire communication

Within a two wire communication, such as I^2C, the command-response pair shall be implemented as the concatenation of 2 frames, one of kind W (write) and the other of kind R (read). In order to allow this implementation, a Device ID shall be defined, to be used in the first byte sent by the host to the module. That device ID shall be coded as:

Tabelle B.1 — Device ID coding for a two-wire implementation

b7	b6	b5	b4	b3	b2	b1	b0
1	1	1	0	0	b1 in Table 3	b0 in Table 3	R/W

An example of the use of this communication interface for emBioAPI_BSPAttach command is:

a) Command:

1) The host sends:

i) START condition

ii) Device ID with W

iii) aa 19 00 08 kk gg rr ... rr xx (see clause 9.1.1)

iv) STOP condition

b) Response:

1) The host sends:

i) START condition

ii) Device ID with R

2) The module sends:

i) aa ee vv vv bb ... bb mm ff ... ff xx xx

3) The host sends

i) STOP condition

B.5 SPI communication

As with the asynchronous serial communication, implementation of emBioAPI using SPI communication interface does not require specific framing or packing. Therefore direct implementation is possible.

Annex C
(informative)

Command exchange examples for several scenarios

C.1 Introduction

This annex covers several scenarios for implementing Embedded BioAPI. For each of these, the commands required by the module, as well as the command exchange for several operations, are identified. At the end of this annex, general sequences are presented for those cases not fitting exactly the scenarios provided.

C.2 Template storage module

An example of this scenario is a store-on-card, verify-off-card module. The module is only able to securely store templates and offer those templates to the terminal for matching. No biometric operations are provided by the module.

C.2.1 Commands accepted by the module

Management commands

1) emBioAPI_BSPAttach
2) emBioAPI_BSPDetach
3) emBioAPI_QueryUnit
4) emBioAPI_ControlUnit
5) emBioAPI_Cancel

Enrolment related commands

1) emBioAPI_DeleteBIR

Biometric reference management commands

1) emBioAPI_DBStoreBIR
2) emBioAPI_DBGetBIR

C.2.2 Command exchange examples

C.2.2.1 Store one template in the module

1) emBioAPI_BSPAttach
2) emBioAPI_QueryUnit
3) If needed, emBioAPI_ControlUnit
4) emBioAPI_DBStoreBIR
5) emBioAPI_BSPDetach

C.2.2.2 Get one template from module

1) emBioAPI_BSPAttach
2) emBioAPI_QueryUnit
3) If needed, emBioAPI_ControlUnit
4) emBioAPI_DBGetBIR
5) emBioAPI_BSPDetach

C.3 On-module biometric comparison

An example of this scenario can be a on-card biometric comparison module. This kind of module does not allow the reading of any template, and does not include enrolment functions. Only biometric templates are sent to the module. Therefore, no biometric process are included, except for the biometric comparison.

C.3.1 Commands accepted by the module

Management commands

1) emBioAPI_BSPAttach
2) emBioAPI_BSPDetach
3) emBioAPI_QueryUnit
4) emBioAPI_ControlUnit
5) emBioAPI_Cancel

Enrolment related commands

1) emBioAPI_DeleteBIR

Biometric process commands

1) emBioAPI_VerifyMatchExternal (optionally)
2) emBioAPI_VerifyMatchInternal

Biometric reference management commands

1) emBioAPI_DBStoreBIR

C.3.2 Command exchange examples

C.3.2.1 Store one template in the module

1) emBioAPI_BSPAttach
2) emBioAPI_QueryUnit
3) If needed, emBioAPI_ControlUnit

4) emBioAPI_DBStoreBIR

5) emBioAPI_BSPDetach

C.3.2.2 On-card biometric comparison

1) emBioAPI_BSPAttach

2) emBioAPI_QueryUnit

3) If needed, emBioAPI_ControlUnit

4) emBioAPI_VerifyMatchInternal

5) emBioAPI_BSPDetach

C.4 Biometric processing module

An example of this scenario can be a module with the biometric algorithms implemented, so that a low-profile terminal can use it as a co-processor. Within this example, lets consider that no template storage or enrolment functionalities are included. Also, no capture capabilities are considered.

C.4.1 Commands accepted by the module

Management commands

1) emBioAPI_BSPAttach

2) emBioAPI_BSPDetach

3) emBioAPI_QueryUnit

4) emBioAPI_ControlUnit

5) emBioAPI_Cancel

Biometric process commands

1) emBioAPI_Process

2) emBioAPI_ProcessVerifyMatchExternal

3) emBioAPI_VerifyMatchExternal

C.4.2 Command exchange examples

C.4.2.1 Obtain the processed sample BIR from a sample

1) emBioAPI_BSPAttach

2) emBioAPI_QueryUnit

3) If needed, emBioAPI_ControlUnit

4) emBioAPI_Process

5) emBioAPI_BSPDetach

C.4.2.2 Compare a sample with a template

1) emBioAPI_BSPAttach
2) emBioAPI_QueryUnit
3) If needed, emBioAPI_ControlUnit
4) emBioAPI_ProcessVerifyMatchExternal
5) emBioAPI_BSPDetach

C.4.2.3 Compare a feature_vector with a template

1) emBioAPI_BSPAttach
2) emBioAPI_QueryUnit
3) If needed, emBioAPI_ControlUnit
4) In case the host needs to calculate the processed sample BIR, emBioAPI_Process
5) emBioAPI_VerifyMatchExternal
6) emBioAPI_BSPDetach

C.5 Sensor

An example of this scenario can be any kind of sensor without any processing capabilities.

C.5.1 Commands accepted by the module

Management commands

1) emBioAPI_BSPAttach
2) emBioAPI_BSPDetach
3) emBioAPI_QueryUnit
4) emBioAPI_ControlUnit
5) emBioAPI_Cancel

Biometric process commands

1) emBioAPI_Capture

C.5.2 Command exchange examples

C.5.2.1 Capture a sample

1) emBioAPI_BSPAttach
2) emBioAPI_QueryUnit

3) If needed, emBioAPI_ControlUnit

4) emBioAPI_Capture

5) emBioAPI_BSPDetach

C.6 System-on-module

An example of this scenario can be a sensor with biometric processing capabilities. No template storage capabilities neither enrolment functions are considered in this example.

C.6.1 Commands accepted by the module

Management commands

1) emBioAPI_BSPAttach

2) emBioAPI_BSPDetach

3) emBioAPI_QueryUnit

4) emBioAPI_ControlUnit

5) emBioAPI_Cancel

Biometric process commands

1) emBioAPI_Capture

2) emBioAPI_CaptureProcess

3) emBioAPI_VerifyExternal

4) emBioAPI_Process

5) emBioAPI_ProcessVerifyMatchExternal

6) emBioAPI_VerifyMatchExternal

C.6.2 Command exchange examples

C.6.2.1 Capture a sample returning a processed sample BIR

1) emBioAPI_BSPAttach

2) emBioAPI_QueryUnit

3) If needed, emBioAPI_ControlUnit

4) emBioAPI_CaptureProcess

5) emBioAPI_BSPDetach

C.6.2.2 Capture a sample, and compare it with a template

1) emBioAPI_BSPAttach
2) emBioAPI_QueryUnit
3) If needed, emBioAPI_ControlUnit
4) emBioAPI_VerifyExternal
5) emBioAPI_BSPDetach

C.6.2.3 Other functionalities

A module of this kind could be also able to cover subsections C.4 and C.5.

C.7 System-on-module with enrolment

With this example the enrolment functions will be described. A module of this kind might be able to perform any kind of the functions provided by emBioAPI.

C.7.1 Commands accepted by the module

Management commands

1) emBioAPI_BSPAttach
2) emBioAPI_BSPDetach
3) emBioAPI_QueryUnit
4) emBioAPI_ControlUnit
5) emBioAPI_Cancel

Enrolment related commands

1) emBioAPI_Capture4Enrol
2) emBioAPI_GetBIR
3) emBioAPI_CreateTemplate
4) emBioAPI_DeleteBIR

Biometric process commands

1) emBioAPI_Capture
2) emBioAPI_CaptureProcess
3) emBioAPI_VerifyExternal
4) emBioAPI_VerifyInternal
5) emBioAPI_Process

6) emBioAPI_ProcessVerifyMatchExternal

7) emBioAPI_ProcessVerifyMatchInternal

8) emBioAPI_VerifyMatchExternal

9) emBioAPI_VerifyMatchInternal

Biometric reference management commands

1) emBioAPI_DBStoreBIR

2) emBioAPI_DBGetBIR

C.7.2 Command exchange examples

C.7.2.1 Create a template through enrolment by capturing samples

1) emBioAPI_BSPAttach

2) emBioAPI_QueryUnit

3) If needed, emBioAPI_ControlUnit

4) emBioAPI_Capture4Enrol

5) emBioAPI_Capture4Enrol (as many times as needed)

6) emBioAPI_CreateTemplate

7) emBioAPI_BSPDetach

C.7.2.2 Create a template through enrolment by receiving samples

1) emBioAPI_BSPAttach

2) emBioAPI_QueryUnit

3) If needed, emBioAPI_ControlUnit

4) emBioAPI_GetBIR

5) emBioAPI_ GetBIR (as many times as needed)

6) emBioAPI_CreateTemplate

7) emBioAPI_BSPDetach

C.8 General examples for Enrolment and Verification

For those implementations that do not fit exactly the above mentioned scenarios, this clause provides guidance for achieving enrolments and verifications.

C.8.1 Enrolment

1) emBioAPI_BSPAttach; // Initialization

2) emBioAPI_Capture4Enrol; // Sample BIR (image) capture

3) emBioAPI_CreateTemplate; // Create and store a biometric reference

4) emBioAPI_BSPDetach; // Shut down the module

C.8.2 Verification (1st example)

In this scenario, sample BIR capture, creation of a processed biometric sample from a captured sample, and biometric comparison of a processed biometric sample and a reference stored in an embedded device (e.g. a smart card) are assumed to be implemented in separate functions, called emBioAPI_Capture, BipAPILite_Process, emBioAPI_VerifyMatchInternal, respectively.

1) emBioAPI_BSPAttach; // Initialization

2) emBioAPI_Capture; // Biometric sample (image) capture

3) emBioAPI_Process; // Create a biometric processed sample BIR

4) emBioAPI_VerifyMatchInternal; // Compare this vector with the stored reference

5) emBioAPI_BSPDetach; // Shut down the module

C.8.3 Verification (2nd example)

In this scenario, sample BIR capture and creation of a processed sample BIR from a captured sample are done in one function emBioAPI_CaptureProcess, while biometric comparison of a processed sample BIR and a reference stored in an embedded device (e.g. a smart card) is done in another function emBioAPI_VerifyMatchInternal.

1) emBioAPI_BSPAttach; // Initialization

2) emBioAPI_CaptureProcess; // Sample BIR (image) capture and processing, returning a processed sample BIR

3) emBioAPI_VerifyMatchInternal; // Compare this vector with the stored reference

4) emBioAPI_BSPDetach; // Shut down the module

C.8.4 Verification (3rd example)

In this scenario, sample BIR capture occurs in emBioAPI_Capture, while creation of a processed sample BIR from a captured sample and biometric comparison of a processed sample BIR and a reference stored in an embedded device (e.g. a smart card) is done in another function emBioAPI_ProcessVerifyMatchInternal.

1) emBioAPI_BSPAttach; // Initialization

2) emBioAPI_Capture; // Sample BIR (image) capture

3) emBioAPI_ProcessVerifyMatchInternal; // Get a processed sample BIR from a sample BIR and compare this vector with the stored reference

4) emBioAPI_BSPDetach; // Shut down the module

C.8.5 Verification (4th example)

In this scenario, sample BIR capture, creation of a processed sample BIR from a captured sample, and biometric comparison of a processed sample BIR and a reference stored in an embedded device (e.g. a smart card) are implemented in one function emBioAPI_VerifyInternal.

1) emBioAPI_BSPAttach; // Initialization

2) emBioAPI_VerifyInternal; // Capture a sample BIR, create a processed sample BIR and compare this vector with the stored reference

3) emBioAPI_BSPDetach; // Shut down the module

ICS 35.040

Price based on 41 pages