INTERNATIONAL STANDARD

ISO/IEC 19784-1

First edition
2006-05-01

Information technology — Biometric application programming interface —

Part 1: BioAPI specification

Technologies de l'information — Interface de programmation d'applications biométriques —

Partie 1: Spécifications BioAPI

Reference number
ISO/IEC 19784-1:2006(E)

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 19784-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 37, *Biometrics*.

ISO/IEC 19784 consists of the following parts, under the general title *Information technology — Biometric application programming interface*:

— *Part 1: BioAPI specification*

— *Part 2: Biometric archive function provider interface*

This is the first ISO/IEC standard on BioAPI. Previous versions were published by ANSI and the BioAPI Consortium. As the last official non-ISO release was designated Version 1.1, this ISO/IEC 19784-1 version is designated Version 2.0. This is to distinguish the versions of BioAPI products in the marketplace.

Introduction

This part of ISO/IEC 19784, the BioAPI specification, provides a high-level generic biometric authentication model suited to most forms of biometric technology. No explicit support for multimodal biometrics is currently provided.

An architectural model is described which enables components of a biometric system to be provided by different vendors, and to interwork through fully-defined Application Programming Interfaces (APIs).

A key feature of the architecture is the BioAPI Framework, which supports calls by one or more application components (provided by different vendors, and potentially running concurrently) using the BioAPI API specification. The BioAPI Framework provides this support by invoking (through a Service Provider Interface, SPI) one or more biometric service provider (BSP) components (provided by different vendors, and potentially running concurrently) which can be dynamically loaded and invoked as required by an application component.

At the lowest level there is hardware or software that performs biometric functions such as capture, matching, or archiving. These parts of the architecture are called BioAPI Units, and can be integral to a BSP or can be supplied as part of a separate BioAPI Function Provider (BFP) component.

Interactions (through the BioAPI Framework) can occur between BSPs from different vendors provided data structures used to record information from the BioAPI Units they access conform to other International Standards, and in particular to ISO/IEC 19794 [5].

The final component of the BioAPI architecture is the recognition that a BSP can provide its biometric services either:

a) by the use of BioAPI Units that are integral to (that is, directly managed by) the BSP, or

b) by invoking, through the BioAPI Function Provider Interface (FPI), one or more BFP components (provided by different vendors) that manage BioAPI Units that are integral to the BFP.

NOTE: A BioAPI Unit may consist of software only, or a combination of software and hardware (e.g., a biometric sensor, archive, or algorithm).

For each type of BioAPI Unit supported by a BSP (or BFP) there may be one or more BioAPI Units of that type which can be dynamically inserted and removed from the system. Insertion and removal generates events that can be signalled (through the BSP and the BioAPI Framework) to an application.

The BioAPI specification covers the basic biometric functions of Enrollment, Verification, and Identification (see Annex C), and includes a database interface to allow an application to manage the storage of biometric records through an archive BioAPI Unit managed by a BSP or BFP. This provides for optimum performance (e.g., when performing the biometric Identification function within a large population) of the archiving and biometric search processes.

The interface to the application provides primitives that allow it to manage the capture of biometric samples from a biometric sensor by accessing the corresponding BioAPI Unit, and the use of those biometric samples for Enrollment (storage in an application-controlled or BSP-controlled BIR database), and subsequent Verification or Identification against those stored records.

This part of ISO/IEC 19784 also specifies the content of a biometric component registry (information about the biometric components that have been installed on the biometric system). It also provides a component registry interface for the management and inspection of that registry.

This part of ISO/IEC 19784 uses the C programming language (see ISO/IEC 9899) to specify the data structures and function calls that form the BioAPI interfaces.

Clause 6 describes the BioAPI architectural model and its components, and the interfaces that are specified between these components.

Clause 7 defines the data structures used in the BioAPI.

Clause 8 defines the function calls initiated by an application and supported by a conforming BioAPI Framework that are either handled internally by the BioAPI Framework (for example enumeration of installed BioAPI components) or mapped to a function provided by a BSP.

Clause 9 defines the function calls supported by a conforming BSP (and invoked by the BioAPI Framework in response to a call from a biometric application).

Clause 10 specifies the form of the biometric component registry and the component registry interface.

Clause 11 defines the handling of events and error returns.

Annex A is normative, and specifies details of conformance requirements and proformas that can be used by the vendor of a BioAPI Biometric Application, Framework, or BSP component to identify those functions and biometric record formats that must be supported.

NOTE: A future International Standard (ISO/IEC 24709) will address conformance testing for this BioAPI specification. [7]

Annex B is normative, and specifies the BioAPI biometric information record (BIR) as a CBEFF Patron Format in accordance with ISO/IEC 19785-1. It provides a description of the biometric record specified in this part of ISO/IEC 19784, together with the platform-independent bit-pattern representation of such a record for storage and transfer.

Annex C is informative, and provides a general tutorial on a number of aspects of the BioAPI specification.

Annex D is informative, and provides example code to illustrate calling sequences and to provide implementation guidance.

Information technology — Biometric application programming interface —

Part 1: BioAPI specification

1 Scope

This part of ISO/IEC 19784 defines the Application Programming Interface (API) and Service Provider Interface (SPI) for standard interfaces within a biometric system that support the provision of that biometric system using components from multiple vendors. It provides interworking between such components through adherence to this part of ISO/IEC 19784 and to other International Standards.

The BioAPI specification is applicable to a broad range of biometric technology types. It is also applicable to a wide variety of biometrically enabled applications, from personal devices, through network security applications, to large complex identification systems.

This part of ISO/IEC 19784 supports an architecture in which a BioAPI Framework supports multiple simultaneous biometric applications (provided by different vendors), using multiple dynamically installed and loaded (or unloaded) biometric service provider (BSP) components and BioAPI Units (provided by other different vendors), possibly using one of an alternative set of BioAPI Function Provider (BFP) components (provided by other vendors) or by direct management of BioAPI Units.

NOTE: Where BioAPI Units are provided by a different vendor fom a BSP, a standardised BioAPI Function Provider Interface (FPI) may be needed. This is outside the scope of this part of ISO/IEC 19784, but is specified by later parts for the different categories of FPI.

This part of ISO/IEC 19784 is not required (and should normally not be referenced) when a complete biometric system is being procured from a single vendor, particularly if the addition or interchange of biometric hardware, services, or applications is not a feature of that biometric system. (Such systems are sometimes referred to as "embedded systems".) Standardisation of such systems is not in the scope of this part of ISO/IEC 19784.

It is not in the scope of this part of ISO/IEC 19784 to define security requirements for biometric applications and biometric service providers.

NOTE: ISO 19092 provides guidelines on security aspects of biometric systems. [3]

The performance of biometric systems (particularly in relation to searches of a large population to provide the biometric identification capability) is not in the scope of this part of ISO/IEC 19784. Trade-offs between interoperability and performance are not in the scope of this part of ISO/IEC 19784.

This part of ISO/IEC 19784 specifies a version of the BioAPI specification that is defined to have a version number described as Major 2, Minor 0, or version 2.0.

NOTE: Earlier versions of the BioAPI specification were not International Standards.

2 Conformance

2.1 Annex A specifies the conformance requirements for BioAPI components claiming conformance to this part of ISO/IEC 19784.

2.2 This part of ISO/IEC 19784 uses the C programming language (see ISO/IEC 9899) to specify the interfaces that it defines. A BioAPI component can conform to this part of ISO/IEC 19784 by the provision or use of that interface with languages other than the C programming language, provided that the component on the other side of such an interface can use the interface through the detailed C programming language specification given in this part of ISO/IEC 19784. (See also clause 7.1.)

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 9834-8, *Information technology — Open Systems Interconnection — Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components*

ISO/IEC 9899:1999, *Programming Languages — C*

ISO/IEC 10646:2003, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*

ISO/IEC 19785-1, *Information technology — Common Biometric Exchange Formats Framework — Part 1: Data element specification*

ISO/IEC 19785-2, *Information technology — Common Biometric Exchange Formats Framework — Part 2: Procedures for the operation of the Biometric Registration Authority*

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 part of ISO/IEC 19784.

4.1
adaptation
template adaptation
use of a BIR produced from a newly captured and verified biometric sample to automatically update or refresh a reference template

NOTE: This procedure is used to minimize the effects of template aging.

4.2
attach session
temporary association between an application, a single BSP, and a set of BioAPI Units that are managed either directly or indirectly by that BSP

4.3
BioAPI component
component of the BioAPI architecture with a defined interface that can be supplied by a separate vendor and which is subject to conformance testing

NOTE: BioAPI components include BioAPI applications, the BioAPI Framework, BSPs, and BFPs.

4.4
BioAPI Function Provider
BFP
component that manages one or more BioAPI Units of a specific category

NOTE 1: Interfaces to BioAPI Function Providers are standardized in subsequent parts of ISO/IEC 19784.

NOTE 2: BFPs are categorized according to the categories of BioAPI Units that they manage (see clause 6.2.4).

4.5
BioAPI Unit
abstraction of a hardware or software resource that is directly managed by a BSP or BFP

NOTE: BioAPI Units are categorized (see clause 6.2.2) and include sensor units, archive units, matching algorithm units and processing algorithm units.

4.6
biometric (adj.)
pertaining to the field of biometrics

4.7
biometric data block
BDB
block of data with a defined format that contains one or more biometric samples or biometric templates

NOTE 1: ISO/IEC 19784 does not support BDB formats that are not an integral multiple of eight bits.

NOTE 2: There is no requirement that a BDB format be self-delimiting.

NOTE 3: Each part of ISO/IEC 19794 standardises one or more BDB formats. Vendor specific formats can also be specified and identified.

NOTE 4: Within BioAPI, the BDB is “opaque” to the application and is therefore sometimes referred to as an opaque biometric data block.

4.8
biometric information record
BIR
data structure containing one or more BDBs, together with information identifying the BDB formats, and possibly further information such as whether a BDB is signed or encrypted (see ISO/IEC 19785-1)

NOTE: This part of ISO/IEC 19784 defines a BIR format (see clause 7.4) that supports only a single BDB. ISO/IEC 19785-1 defines a more general BIR format that supports multiple BDBs within the BIR, and the above definition is used in common by the two International Standards. When the term BIR is used in this part of ISO/IEC 19784, it normally refers to the specific BIR format defined by BioAPI (see Annex B), not to an arbitrary BIR. The term BioAPI BIR is used where clarity is needed.

4.8.1
reference BIR
BIR whose BDB(s) contain one or more biometric templates

4.8.2
sample BIR
BIR whose BDB(s) contain only biometric samples that are not templates

4.9
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.

4.9.1
biometric template
biometric sample or combination of biometric samples that is suitable for storage as a reference for future comparison

4.9.2
intermediate biometric sample
biometric sample obtained by processing a raw biometric sample, intended for further processing

4.9.3
processed biometric sample
biometric sample suitable for comparison

4.9.4
raw biometric sample
biometric sample obtained directly from a biometric sensor

NOTE: The formats for raw biometric samples are not currently standardised, and depend on the nature of the biometric device and the vendor of that device. They may in the future be standardised as part of the standardisation of specific biometric devices.

4.9.5
reference template
biometric template that has been stored

4.10
biometric sensor
biometric hardware used to capture raw biometric samples from a subject

NOTE: The term 'biometric device' is used interchangeably with this term.

4.11
biometric service provider
BSP
component that provides biometric services to an application through a defined interface by managing one or more BioAPI Units directly, or through interfaces to BioAPI Function Providers

4.12
biometrics (noun)
automated recognition of individuals based on their behavioural and biological characteristics

4.13
callback
mechanism by which a component that exposes an API invokes a function within a component that uses that API, where the address of that function has been previously passed as an input parameter of an API function call

NOTE: This mechanism enables a BioAPI component to communicate with another BioAPI component other than by invoking an API function, usually in response to an event or interrupt.

4.14
component registry
information maintained by the BioAPI Framework concerning the BioAPI components that are available on a biometric system

4.15
encrypt
encryption
(reversible) transformation of data by a cryptographic algorithm to produce ciphertext; that is, to hide the information content (protect the confidentiality) of the data

NOTE 1: Encryption algorithms consist of two processes: encryption (or encipherment) which transforms plaintext into ciphertext, and decryption (or decipherment) which transforms ciphertext to plaintext.

NOTE 2: Encryption may be used for either security or privacy reasons.

4.16
enrollment
process of collecting one or more biometric samples from an individual, and the subsequent construction of a biometric reference template which can then be used to verify or determine the individual's identity

NOTE: The reference template would normally be stored by a biometric application, a BSP supporting an archive BioAPI Unit, or both.

4.17
False Match Rate
FMR
measure of the probability that a biometric matching process will incorrectly identify an individual or will fail to reject an impostor

NOTE 1: Within BioAPI, FMR is used as a means of specifying scores and thresholds (see clause C.4).

NOTE 2: Historically, False Acceptance Rate (FAR) has also been used with a similar definition, but FMR is the preferred term in International Standards. Similarly for False Rejection Rate (FRR), as opposed to the preferred False Non-Match Rate (FNMR).

4.18
handle
parameter returned by a BioAPI function (A say) that can be used by the BioAPI application in a subsequent function call to identify a BioAPI component or data element within the component A

NOTE: Types of handles include:

BIR_Handle, generated by a BSP to select or access a BIR within that BSP.

BSP Attach Session Handle, for an attach session.

DB_Handle, generated by a BSP to select or access a BIR database controlled by that BSP.

4.19
identify
identification
one-to-many process of comparing a submitted biometric sample against a reference population to determine whether the submitted biometric sample matches any of the reference templates in that reference population in order to determine the identity of the enrollee whose template was matched

NOTE: This is often called an "identification match" or "identifymatch".

4.20
match
matching
one-to-one process of comparing a submitted biometric sample against a single biometric reference template and scoring the level of similarity.

NOTE 1: An accept or reject decision would then normally be based upon whether this score exceeds a given threshold.

NOTE 2: Matching algorithms and their effect on False Match Rate and False Non-Match Rate scores are currently not standardised.

NOTE 3: See also identify (4.19) and verify (4.28).

4.21
payload
data, provided at the time of enrollment and associated with a reference template, which can be released upon a successful biometric verification.

NOTE: Examples of payloads include user names, accounts, passwords, cryptographic keys, or digital certificates (see clause C.5).

4.22
score
scoring
value indicating the degree of similarity or correlation between a biometric sample and a biometric reference template

4.23
security block
SB
block of data with a defined format that contains security information (for example, related to encryption or integrity) related to a BIR (see ISO/IEC 19785-1)

4.24
self-contained device
combination device which includes a biometric sensor and all or part of the BSP functionality

NOTE: A self-contained device may include the ability to not only capture a biometric, but also to process, match, and/or store it. This functionality is typically implemented in hardware/firmware.

4.25
signature
digital signature
data appended to, or a cryptographic transformation of, a data unit that allows the recipient of the data unit to prove the origin and integrity of the data unit and protect against forgery, e.g. by the recipient

NOTE: Digital signatures may be used for purposes of authentication, data integrity, and non-repudiation

4.26
threshold
predefined value which establishes the degree of similarity or correlation (that is, a score) necessary for a biometric sample to be deemed a match with a biometric reference template

4.27
universally unique identifier
UUID
128-bit value generated in accordance with ISO/IEC 9834-8 and providing unique values between systems and over time

4.28
verify
verification
one-to-one process of comparing a single submitted biometric sample against a biometric reference template to determine whether the submitted biometric sample matches the reference template

NOTE: This is often called a "verification match" or "verifymatch".

5 Symbols and abbreviated terms

API – Application Programming Interface

BDB – Biometric Data Block

BFP – BioAPI Function Provider

BIR – Biometric Information Record

BSP – Biometric Service Provider

CBEFF – Common Biometric Exchange Formats Framework

FMR – False Match Rate

FPI – Function Provider Interface

GUI – Graphical User Interface

ID – Identity/Identification/Identifier

MOC – Match on Card

PID – Product ID

SB – Security Block

SBH – Standard Biometric Header

NOTE: This term and abbreviation is imported from ISO/IEC 19785-1.

SPI – Service Provider Interface

UUID – Universally Unique Identifier

6 The BioAPI architecture

6.1 The BioAPI API/SPI Architectural Model

6.1.1 The BioAPI incorporates an API/SPI model, illustrated in Figure 1.

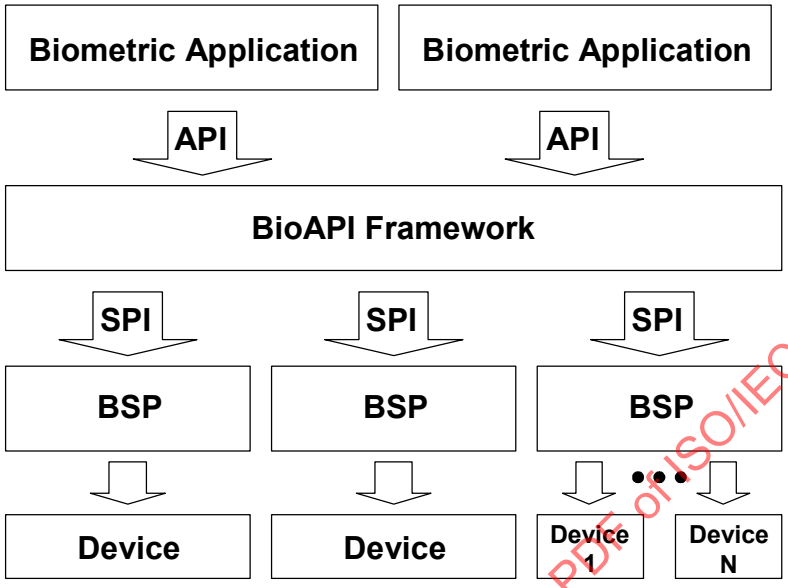

Figure 1 — BioAPI API/SPI Model

NOTE: The structure below the SPI is simplified in Figure 1. The options (and standardized interfaces) for the internal structure of the BSPs are presented in Figure 2.

6.1.2 A BioAPI system consists of BioAPI components providing standardized interfaces. Figure 1 shows interactions between three sorts of BioAPI components: the BioAPI Framework, applications, and monolithic BSPs.

NOTE: Standardized interfaces to allow the functions of a BSP to be provided by independent BioAPI components are described in 6.2.

6.1.3 The API specifies the interface between the BioAPI Framework and a biometric application. The application is written to invoke the functions in the API specification (see clause 8). The BioAPI Framework supports the functions in the API specification.

6.1.4 The SPI provides the interface between the BioAPI Framework and BSPs. The BioAPI Framework invokes the functions in the SPI specification (see clause 9). The BSP supports the functions of the SPI specification.

6.1.5 The BioAPI Framework provides for management of BSPs and for the mapping of function calls from an API function to an SPI function addressed to the appropriate BSP.

6.1.6 An application can access the functionalities of a BSP (through the BioAPI Framework) only after the BSP has been loaded and attached (see clauses 8.1.5 and 8.1.7). When the application no longer requires the use of the BSP, it is detached and unloaded (see clauses 8.1.6 and 8.1.8).

6.1.7 An application may load (and attach to) more than one BSP at a time. A BSP may be attached to more than one application at a time.

NOTE: Interactions between an application and an attached BSP are normally independent of other interactions between that application and other BSPs, or between that BSP and other applications, except when there is contention for a physical device managed by the BSP.

6.1.8 The functions specified in this part of ISO/IEC 19784 support the presence in a biometric system of:

a) a singleBioAPI Framework with an associated component registry; and

b) dynamic execution and termination of multiple simultaneous biometric applications interworking with that BioAPI Framework; and

c) the dynamic installation, deinstallation (and associated loading and unloading) of multiple BSPs interworking with that BioAPI Framework; and

d) signaling from a BSP to the BioAPI Framework (and hence to running biometric applications) of events related to the dynamic connection and disconnection of BioAPI Units (see clause 6.5) managed by that BSP.

NOTE: It is expected that the most common use of this API will be for a single application to control one BSP with at most one BioAPI Unit of each biometric category accessed through that BSP. However, support for an application to access multiple BSPs, each capable of managing multiple BioAPI Units (but only one of each category for a given attached session) will facilitate such applications as physical access control, especially when networked devices are employed.

6.2 The BioAPI BSP Architectural Model

6.2.1 BioAPI Units are the abstraction of biometric devices and are the basic building blocks presented to an application by a BSP. They encapsulate and hide software and hardware resources (of various types – sensor device, archive device, etc.).

6.2.2 Each BioAPI Unit models or encapsulates a single (or none) piece of hardware and any necessary software. The following are the currently defined categories of BioAPI Unit:

- Sensor unit,
- Archive unit,
- Matching algorithm unit, and
- Processing algorithm unit

NOTE: It is likely, but not required, that the first two units will be associated with detachable hardware, and that the last two units will not have any associated hardware.

6.2.3 A BioAPI Unit may be managed internally by a BSP (direct management of the BioAPI Unit) or may be managed by an associated BioAPI Function Provider (BFP) (indirect management of the BioAPI Unit). The BioAPI Function Provider Interfaces (FPIs) are specified in subsequent parts of ISO/IEC 19784.

6.2.4 A BFP can manage multiple BioAPI Units of a given category (but not more than one in any attach session for a given application). BFPs are categorized according to the categories of the BioAPI Units that they can manage. The following are the current categories of BFPs:

- Sensor BFP,
- Archive BFP,
- Matching algorithm BFP, and
- Processing algorithm BFP.

6.2.5 The BioAPI FPIs for each of the categories of BFP are specified in other parts of ISO/IEC 19784. All FPIs support access to one or more (but not more than one in any attach session for a given application) BioAPI Units in the addressed BFP.

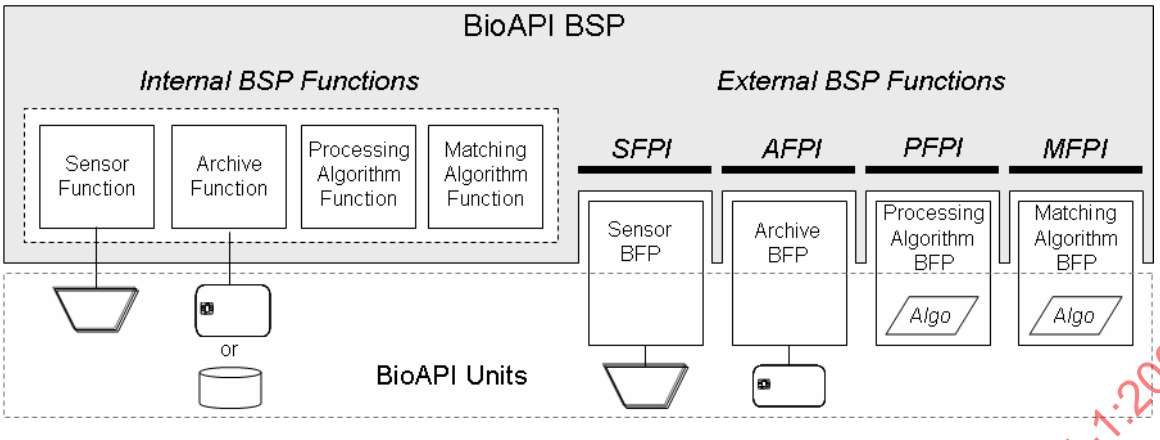

Figure 2 — Illustration of BSP architecture

6.2.6 A BSP that supports multiple BioAPI Units (either directly managed or through BFPs) supports the SPI interface that enables an application to select a particular BioAPI Unit (one of each category) for an attach session. The application may indicate BioAPI_DONT_CARE for the BioAPI Unit of a particular category, in which case the BSP selects the BioAPI Unit to be used.

6.2.7 If the implementer of a BSP claims (see clause 7.47) to support a particular category of BioAPI Unit, then the BSP is required either to be able to manage a BioAPI Unit of that category directly or to be able to interact with a BFP of a corresponding category through an associated (standardized) FPI.

6.2.8 There is a BioAPI API function and corresponding SPI function (and in some cases FPI functions) (see clauses 8.1.12 and 9.3.1.7) that allow an application to send/request control/status information to/from a BioAPI Unit using the BioAPI API and SPI (and FPI). This is in addition to the normal biometric operations. The parameters of the control functions are not standardized. If either the BSP or BFP (if involved) does not support this control function, an error is returned (see clause 6.6 for further details).

6.2.9 When a BSP attach session is established, at most one BioAPI Unit of each category is selected (see clause 8.1.7).

NOTE: This architecture requires that the FPI interfaces allow the BSP to pass to a BFP the identification (provided through the BioAPI and the BioSPI interface) of the precise BioAPI Unit that is to be used. This implies the particular camera, scanner, storage device, etc. that is to be used. Subsequent parts of ISO/IEC 19784 define the necessary functions of the FPI for each category of BFP.

6.3 The Component Registry

6.3.1 The component registry contains information about installed BSPs and BFPs.

6.3.2 In the BioAPI model, there is a single BioAPI Framework and a single associated component registry in a biometric system.

NOTE: In a real computer system, there may be multiple component registries, either supported by the same (shared) BioAPI Framework code, or by different BioAPI Framework code (perhaps old and new versions of the Framework). This is modelled as multiple biometric systems in a single real computer system.

6.3.3 It is a BioAPI requirement that there shall be no interaction or interference between different biometric systems implemented in a single, real computer system.

NOTE: Where a real computer system contains multiple biometric systems, the possible sharing of code and the means by which an application is bound to one or other biometric system is an implementation matter.

6.3.4 The following information can be obtained by an application from BioAPI Framework functions that return information in the component registry (see clauses 8.1.3, 8.1.4, and 8.1.10):

a) information about the BioAPI Framework itself,

b) details of all installed BSPs, and

c) details of all installed BFPs.

NOTE: Information about installed BFPs can also be obtained by a BSP through a callback mechanism.

6.3.5 The following information can be obtained by an application from BioAPI Framework functions that are passed via the SPI to a specific BSP (see clauses 8.1.11 and 8.1.9).

a) details of all the installed BFPs that are supported by that BSP, and

b) details of all BioAPI Units that are in the inserted state and are accessible through that BSP (either directly or through a supported BFP).

6.3.6 The information listed in clauses 6.3.4 and 6.3.5 can be obtained by an application using functions that can be called at the following times:

a) information about the framework itself can be obtained at any time after ***BioAPI_Init*** (see clause 8.1.3).

b) details of all installed BSPs can be obtained at any time after ***BioAPI_Init*** (see clause 8.1.4).

c) details of all installed BFPs can be obtained at any time after ***BioAPI_Init*** (see clause 8.1.10).

d) details of all the installed BFPs that are supported by each BSP can be obtained at any time after ***BioAPI_BSPLoad*** (see clause 8.1.11) of that BSP.

NOTE 1: Multiple BSPs can be simultaneously loaded by an application.

NOTE 2: This implies that when a ***BioAPI_BSPLoad*** occurs, the BSP uses the FPI to load all the BFPs that it is capable of using.

e) details of all BioAPI Units that are in the inserted state and that are accessible through each BSP (either directly or through a supported BFP) can be obtained (by quoting the BSP UUID) at any time after ***BioAPI_BSPLoad*** (see clause 8.1.9) of that BSP.

6.4 BSP and BFP Installation and De-installation

6.4.1 On installation of a BSP or BFP, a UUID is required (a parameter of ***BioAPI_Util_InstallBSP*** and ***BioAPI_Util_InstallBFP***) to identify the BSP or BFP. If there is an attempt to install a BSP or BFP with the same UUID as an already installed BSP or BFP, then the install shall fail (return an error). If the same BSP or BFP is installed on multiple biometric systems, then it may or may not (but should) have the same UUID on all systems. The UUID is required to be distinct across all BSPs and BFPs in a single biometric system.

6.4.2 The installation of a BSP is done by calling a BioAPI Framework function (see clause 10.2.1) and supplying details of the BSP (the BSP schema, specified in clause 7.16).

NOTE: This function call is standardized, but its invocation is typically from an installation wizard application rather than a normal biometric application.

6.4.3 When a BSP is newly installed, it will typically be activated by the installation process using implementation-dependent mechanisms and can then use a callback mechanism (see clause 9.2.2) in order to obtain information from the Framework about installed BFPs.

NOTE: This function can be called at any time to enable a BSP to update any internal information that it might maintain. The format and content (and even the existence) or such internal information is not standardized and is purely a matter for the BSP.

6.4.4 After successful installation or de-installation of a BSP or BFP, the next call from any application or BSP retrieving information from the component registry shall return correct information related to a newly installed BSP or BFP and no information about an uninstalled BSP or BFP (see clauses 8.1.4 and 8.1.10).

6.4.5 There are no mechanisms to inform an installed BSP (whether currently in use or not) about the installation of new BFPs or the de-installation (see also clause 6.4.6) of existing BFPs. Responsibility for determining what BFPs are installed resides solely with the BSPs using the BFP Enumeration Handler callback (see clause 9.2.2).

6.4.6 There is a BioAPI API function (see clause 10.2.1) that can be called by an application (for example, a de-installation wizard) to inform the BioAPI Framework that a BSP has been uninstalled. The BioAPI Framework will update the component registry. There are no standardized functions to inform BSP instances in the working memory of running applications about the de-installation of a BFP that it is using, and the effect of uninstalling a BFP that is in use by a BSP is implementation-dependent.

NOTE: The de-installation of a BFP will not normally uninstall related hardware or drivers for BioAPI Units, as such drivers and hardware may also be under the control of other installed BFPs. This area is outside the standardization of BioAPI, and is again implementation-dependent.

6.5 BSP Load and BioAPI Unit Attachment

6.5.1 The actions of an application wishing to perform biometric operations are (in order):

a) initialize access to the BioAPI Framework (see clause 8.1.1);

b) identify and load one or more BSPs (see clauses 8.1.4 and 8.1.5), optionally specifying a callback event handler to receive callbacks when defined events (see clause 7.26) occur related to that BSP (for example, insertion or removal of a BioAPI Unit that can be accessed through that BSP). When callback event handlers are specified, the application will receive notifications about all insertions or removals that the associated BSPs become aware of;

c) attach a BSP together with at most one BioAPI Unit of each category (either directly or indirectly accessed by that BSP);

NOTE: 'attach a BSP' means 'establish an attach session to a BSP'.

d) following the attach, the other BioAPI function calls can be made with reference to the attached BSP, and will be handled by the attached BSP using the identified BioAPI Units of the required category.

NOTE 1: It is possible for an application to establish multiple simultaneous attach sessions with different BSPs (or even with the same BSP).

NOTE 2: Annex D provides sample calling sequence code.

6.5.2 Loading a BSP (see clause 8.1.5) enables the application to subsequently obtain full information about the BioAPI Units that can be accessed (directly or indirectly) through that BSP, either through a Framework query or by a callback notification from the BSP (or both).

6.5.3 A BioAPI Unit can be used if and only if any hardware or software resource that it depends on is currently available in the system. This is modeled by saying that the BioAPI Unit is in the inserted state.

6.5.4 When an application attaches a BSP it may indicate that the BioAPI Unit to be selected for any particular BioAPI Unit category should be determined by the BSP. This is called "selecting a default BioAPI Unit using BioAPI_DONT_CARE".

NOTE: The only difference between selecting a specific BioAPI Unit (that is in the inserted state) of a given category and selecting the default BioAPI Unit using BioAPI_DONT_CARE is that the decision on which BioAPI Unit to use is taken by the BSP. There is no other difference.

6.5.5 The event notification about each inserted BioAPI Unit includes its schema.

NOTE: It is possible that the physical insertion of some piece of hardware (for example, a smartcard that can support both archiving and matching) might produce two separate and different event notifications. The application may not be able to relate these two events to the same physical device.

6.5.6 On ***BioAPI_BSPAttach***, the application is required to select at most one BioAPI Unit of each category that is currently in an "inserted" state (or to select BioAPI_DONT_CARE) and is managed by that BSP or by an associated BFP. The BSP then either accesses that BioAPI Unit (for directly managed BioAPI Units), or else interacts with the associated BFP in order to access that BioAPI Unit.

6.5.7 The information content of removal notification includes (see clause 7.28):

a) The BioAPI Unit ID
b) Event type (remove)
c) Callback context

6.5.8 The occurrence of a removal event notification for a BioAPI Unit that forms part of the set of BioAPI Units used in an attach session requires that the application issue no further function calls other than ***BioAPI_BSPDetach*** and ***BioAPI_GetBIRFromHandle***.

6.6 Controlling BioAPI Units

6.6.1 A function (see clauses 8.1.12 and 9.3.1.7) is available in the BioAPI API, SPI, and FPIs that enable an application to control a BioAPI Unit through a BSP. A BSP is not required to support this function, and identifies such support in its schema (see clauses 7.16 and 7.46).

6.6.2 This function contains specific control codes, buffer contents, and return values, but the format and meaning of these are identified by a UUID and may be vendor specific. There are no standardized UUIDs for this function in this version of this part of ISO/IEC 19784.

6.7 BIR Structure and Handling

6.7.1 BIR Structure

The BIR that crosses the API and SPI is a C data structure stored in memory using pointers to various elements. The BIR structure recommended for storage and transfer between computer systems is a serialization of that C data structure and is shown graphically in Figure 3 below. In addition to the fields specified in the C structure, length fields are included in the recommended storage and transfer format.

NOTE: Annex B specifies the BioAPI Patron Format which is recommended for storage and transmission of the BioAPI BIR. Representation of the BioAPI BIR in the API and SPI uses the datastructure specified in clause 7.4.

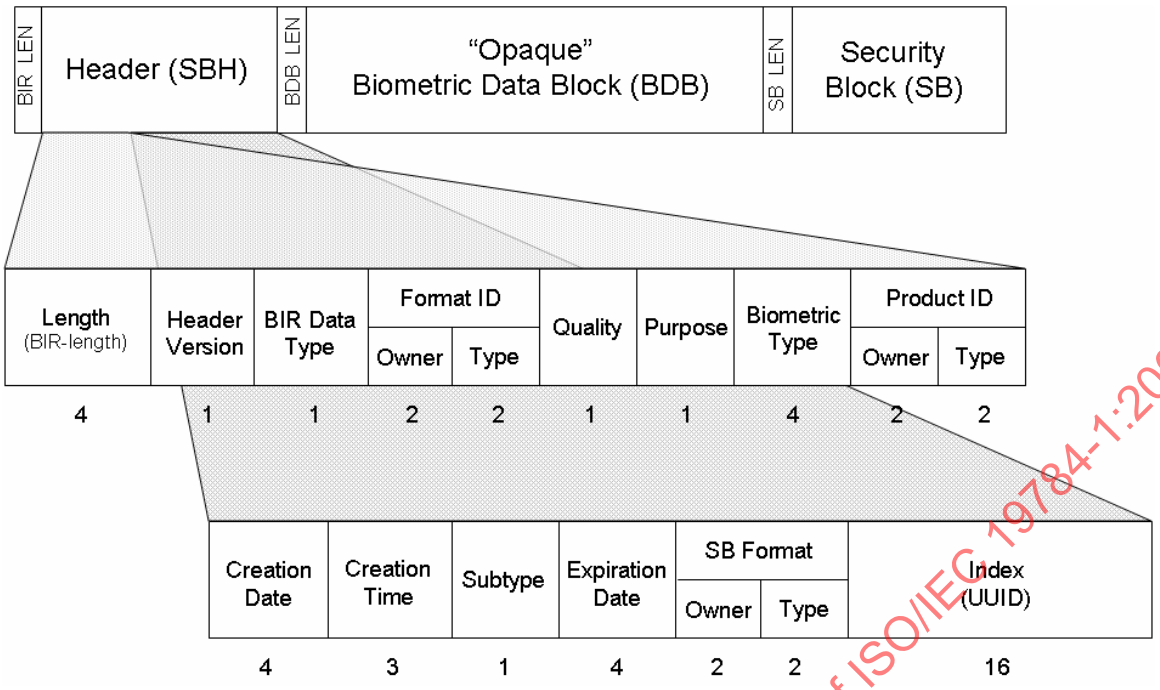

Figure 3 — Biometric information record (BIR)

The Standard Biometric Header (SBH) contains information describing the contents of the following BDB.

The BDB contains the biometric sample whose format is defined by the Format ID field of the SBH. This may be a standard or proprietary format.

NOTE: Standard BDB formats are defined in ISO/IEC 19794.

The security block (SB) is optional (its absence is indicated by a zero length field), and contains parameters associated with the signing and/or encryption of the BIR. The format of the SB is determined by the SB Format field of the SBH. The BIR Data Type indicates whether the BIR is signed and/or the BDB is encrypted (see clause 7.9).

The first length field is the length of the entire BIR, not including this length field itself. The second length field, which is located after the header and before the BDB, is the length of the BDB (not including the length field itself). The third length field, which is located after the BDB and before the SB, is the length of the SB (not including the length field itself).

6.7.2 BIR Data Handling

When a BSP creates a new BIR, it returns a "handle" to it. Most local operations can be performed without moving the BIR out of the BSP. (BIRs can be quite large, so this is a performance advantage.) However, if the application needs to manage the BIR (either to store it in an application database, to pass it to another BSP, or to send it to a server for verification/identification, possibly via the Framework using the future International Standard ISO/IEC 24708 [6]), it can acquire the BIR using the handle (in the function ***BioAPI_GetBIRFromHandle***). If just the header information is needed, it can be retrieved using ***BioAPI_GetHeaderFromHandle***.

Whenever an application needs to provide a BIR as input to a BioAPI function, it can be done in one of three ways:

a) by reference to its handle (if it is in the BSP being invoked);

b) by reference to its key value (UUID) in an open BIR database (controlled by the BSP being invoked); or

c) by supplying the BIR itself using the C structure BioAPI_BIR.

7 BioAPI types and macros

7.1 BioAPI

Definition of the BioAPI calling conventions.

```
#ifdef WIN32
#define BioAPI __stdcall
#else
#define BioAPI
#endif
```

NOTE: Interworking of C programs from different vendors in general depends on choices made (sometimes selectable by statements in the C header) on the in-core representation (for example, padding between data elements, parameter passing mechanisms, and use of registers or stacks). On many operating systems, there is a well understood default for such choices that is in common use, which should be used. Where there is no recognized default, the option that represents data structures without padding between data elements should be used.

7.2 BioAPI_BFP_LIST_ELEMENT

7.2.1 Identifies a BFP, giving its category and UUID. A list is returned by a BSP when queried for the installed BFPs that it supports.

```
typedef struct bioapi_bfp_list_element   {
        BioAPI_CATEGORY  BFPCategory;
        BioAPI_UUID  BFPUuid;
    } BioAPI_BFP_LIST_ELEMENT;
```

7.2.2 Definitions

BFPCategory – Defines the category of the BioAPI Unit that the BFP supports.

BFPUuid – UUID of the BFP in the component registry.

7.3 BioAPI_BFP_SCHEMA

7.3.1 Information about a BFP, maintained in the component registry.

```
typedef struct bioapi_bfp_schema  {
        BioAPI_UUID  BFPUuid;
        BioAPI_CATEGORY  BFPCategory;
        BioAPI_STRING  BFPDescription;
        uint8_t *Path;
        BioAPI_VERSION  SpecVersion;
        BioAPI_STRING  ProductVersion;
        BioAPI_STRING  Vendor;
        BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *BFPSupportedFormats;
        uint32_t  NumSupportedFormats;
        BioAPI_BIR_BIOMETRIC_TYPE  FactorsMask;
        BioAPI_UUID  BFPPropertyID;
        BioAPI_DATA BFPProperty;
    }BioAPI_BFP_SCHEMA;
```

7.3.2 Definitions

BFPUuid – UUID of the BFP.

BFPCategory – Category of the BFP identified by the BFPUuid.

BFPDescription – A NUL-terminated string containing a text description of the BFP.

Path – A pointer to a NUL-terminated string containing the path of the BFP file, including the filename. The path may be a URL. This string shall consist of ISO/IEC 10646 characters encoded in UTF-8 (see ISO/IEC 10646:2003, Annex D).

NOTE: When BioAPI_BFP_SCHEMA is used within a function call, the component that receives the call allocates the memory for the *Path* schema element and the calling component frees the memory.

SpecVersion – Major/minor version number of the FPI specification to which the BFP was implemented.

NOTE: FPI specifications will be issued as subsequent, additional parts of ISO/IEC 19784.

ProductVersion – The version string of the BFP software.

Vendor – A NUL-terminated string containing the name of the BFP vendor.

BFPSupportedFormats – A pointer to an array of BioAPI_BIR_BIOMETRIC_DATA_FORMAT structures specifying the supported BDB formats.

NumSupportedFormats – Number of supported formats contained in BFPSupportedFormats.

FactorsMask – A mask which indicates which biometric types are supported by the BFP.

BFPPropertyID – UUID of the format of the following BFP property.

BFPProperty – Address and length of a memory buffer containing the BFP property. The format and content of the BFP property can either be specified by a vendor or can be specified in a related standard.

7.4 BioAPI_BIR

7.4.1 A container for biometric data. A BioAPI_BIR consists of a BioAPI_BIR_HEADER, a BDB, and an optional SB. The BDB may contain raw sample data, partially processed (intermediate) data, or completely processed data. The BioAPI_BIR may be used to enroll a user (thus being stored persistently), or may be used to verify or identify a user (thus being used transiently).

7.4.2 The BDB and SB are an integral number of octets and are of variable length, up to 2**32-1 octets. If the SB contains a signature, it is calculated on the combined BioAPI_BIR_HEADER and BDB.

```
typedef struct bioapi_bir {
       BioAPI_BIR_HEADER  Header;
       BioAPI_DATA  BiometricData;
       BioAPI_DATA  SecurityBlock;       /* SecurityBlock.Data=NULL if no SB */
} BioAPI_BIR;
```

NOTE 1: The BioAPI BIR contains the information needed for a CBEFF Patron Format as defined in the Common Biometric Exchange Formats Framework (CBEFF), ISO/IEC 19785-1. See Annex B for the specification of the BioAPI BIR Patron Format, including internetworking and storage formats.

NOTE 2: The format of the BiometricData and SecurityBlock are determined by the BioAPI_BIR_BIOMETRIC_DATA_FORMAT and BioAPI_BIR_SECURITY_BLOCK_FORMAT elements of the BioAPI_BIR_HEADER respectively.

NOTE 3: CBEFF (ISO/IEC 19785-1) allows the possibility of different BIR formats from that supported by BioAPI. Conversion between the BioAPI BIR format and other BIR formats is specifed in ISO/IEC 19785-1.

7.5 BioAPI_BIR_ARRAY_POPULATION

An array of BIRs, generally used during identification operations (as input to ***BioAPI_Identify*** or ***BioAPI_IdentifyMatch*** as part of BioAPI_IDENTIFY_POPULATION).

```
typedef struct bioapi_bir_array_population {
    uint32_t NumberOfMembers;
    BioAPI_BIR   *Members;    /* A pointer to an array of BIRs */
} BioAPI_BIR_ARRAY_POPULATION;
```

7.6 BioAPI_BIR_BIOMETRIC_DATA_FORMAT

Defines the format of the data contained within the "opaque" biometric data block (BDB) of the BioAPI BIR, BiometricData element.

```
typedef struct bioapi_bir_biometric_data_format {
    uint16_t FormatOwner;
    uint16_t FormatType;
} BioAPI_BIR_BIOMETRIC_DATA_FORMAT;
```

FormatOwner values are assigned and registered by the CBEFF Registration Authority (see ISO/IEC 19785-2). Format Type is assigned by the Format Owner and may optionally be registered.

NOTE 1: The BioAPI BIR Biometric Data Format corresponds to a combination of the "CBEFF_BDB_format_owner" and "CBEFF_BDB_format_type" in ISO/IEC 19785-1.

NOTE 2: This structure is primarily used within a BIR header; however, it is also used as an input parameter for functions that capture biometric data.

7.7 BioAPI_BIR_BIOMETRIC_PRODUCT_ID

Provides the product identifier (PID) for the BSP that generated the BDB in the BIR (the BiometricData element).

```
typedef struct bioapi_bir_biometric_product_ID {
    uint16_t ProductOwner;
    uint16_t ProductType;
} BioAPI_BIR_BIOMETRIC_PRODUCT_ID;

    #define BioAPI_NO_PRODUCT_OWNER_AVAILABLE       (0x0000)
    #define BioAPI_NO_PRODUCT_TYPE_AVAILABLE        (0x0000)
```

The condition NO VALUE AVAILABLE shall be indicated by setting the value of all components to zero. This value shall be used only for BIRs that are not originally generated by a BioAPI BSP, but originate from another source and have been transformed into a BioAPI BIR. BSPs shall not use this value.

Product Owner values are assigned and registered by the CBEFF Registration Authority as Biometric Organization identifiers (see ISO/IEC 19785-2). Product Type is assigned by the Product Owner and may optionally be registered.

NOTE 1: Product IDs are analogous to Format IDs and the registration process is the same. A single vendor can register for one Format/Product Owner value (a Biometric Organization identifier) that can be used in both fields.

NOTE 2: The BioAPI BIR Biometric Product ID corresponds to the "CBEFF_BDB_product_owner" and "CBEFF_BDB_product_type" in ISO/IEC 19785-1.

7.8 BioAPI_BIR_BIOMETRIC_TYPE

A mask that describes the set of biometric types (factors) contained within a BioAPI BIR or supported by a BSP.

```
typedef uint32_t BioAPI_BIR_BIOMETRIC_TYPE;

    #define BioAPI_NO_TYPE_AVAILABLE          (0x00000000)
    #define BioAPI_TYPE_MULTIPLE              (0x00000001)
    #define BioAPI_TYPE_FACIAL_FEATURES       (0x00000002)
    #define BioAPI_TYPE_VOICE                 (0x00000004)
    #define BioAPI_TYPE_FINGERPRINT           (0x00000008)
    #define BioAPI_TYPE_IRIS                  (0x00000010)
    #define BioAPI_TYPE_RETINA                (0x00000020)
    #define BioAPI_TYPE_HAND_GEOMETRY         (0x00000040)
    #define BioAPI_TYPE_SIGNATURE_DYNAMICS    (0x00000080)
    #define BioAPI_TYPE_KEYSTOKE_DYNAMICS     (0x00000100)
    #define BioAPI_TYPE_LIP_MOVEMENT          (0x00000200)
    #define BioAPI_TYPE_THERMAL_FACE_IMAGE    (0x00000400)
    #define BioAPI_TYPE_THERMAL_HAND_IMAGE    (0x00000800)
    #define BioAPI_TYPE_GAIT                  (0x00001000)
    #define BioAPI_TYPE_OTHER                 (0x40000000)
    #define BioAPI_TYPE_PASSWORD              (0x80000000)
```

NOTE 1: BioAPI_TYPE_MULTIPLE is used to indicate that the biometric samples contained within the BDB (BIR BiometricData) include biometric samples from more than one type of biometric sensor unit (e.g., fingerprint and facial data). Location of the individual samples within the BDB is specified by the Format Owner and identified by the value of the Format Type.

NOTE 2: The condition NO VALUE AVAILABLE is indicated by setting the value to zero. BIRs that are not originally created by BioAPI BSPs should use this value when transformed into a BioAPI BIR if Biometric Type information is not available in the original source record. Transformed BIRs whose biometric type does not correspond to one of the defined types shall use the value for OTHER.

NOTE 3: The BioAPI BIR Biometric Type corresponds to the “CBEFF_BDB_biometric_type” in ISO/IEC 19785-1.

7.9 BioAPI_BIR_DATA_TYPE

7.9.1 This data type is used for three different purposes:

a) it identifies the type of biometric sample (raw, intermediate, or processed) that is contained in the BDB;

b) it identifies whether the BioAPI BIR is encrypted and/or signed;

c) it identifies whether or not an index value is included as part of the BIR header.

NOTE: If the BIR has been encrypted by a BSP, it may not be readable by an application or by another BSP.

7.9.2 Exactly one of the three flags "raw", "intermediate" and "processed" shall be set. If a BIR carrying a BIR data type with multiple flags set is passed to the BioAPI Framework in a parameter of a function call, a BioAPIERR_INVALID_BIR error shall be returned.

NOTE: BIRs that are not originally created by a BioAPI BSP but have been transformed from another data format and for which sample type information is not available may not have set a flag. (BioAPI BSPs set one of the three flags.)

7.9.3 Each of the flags "encrypted" and "signed" may or may not be set.

7.9.4 The 'index' flag shall be set if an index is present in the BIR header and not set if no index is present in the BIR header.

```
typedef uint8_t BioAPI_BIR_DATA_TYPE;

    #define BioAPI_BIR_DATA_TYPE_RAW              (0x01)
    #define BioAPI_BIR_DATA_TYPE_INTERMEDIATE     (0x02)
    #define BioAPI_BIR_DATA_TYPE_PROCESSED        (0x04)
    #define BioAPI_BIR_DATA_TYPE_ENCRYPTED        (0x10)
    #define BioAPI_BIR_DATA_TYPE_SIGNED           (0x20)
    #defineBioAPI_BIR_INDEX_PRESENT               (0x80)
```

NOTE: The BioAPI BIR Data Type corresponds to a combination of the "CBEFF_BDB_processed_level", "CBEFF_BDB_encryption_options", and "CBEFF_BIR_integrity_options" in ISO/IEC 19785-1.

7.10 BioAPI_BIR_HANDLE

A handle to refer to a BioAPI BIR that exists within a BSP.

NOTE: A handle that identifies a BIR is a positive non-zero value. Other values of BioAPI_BIR_HANDLE are reserved for exception indications (currently only -1 and -2).

```
typedef  int32_t BioAPI_BIR_HANDLE;

    #define BioAPI_INVALID_BIR_HANDLE             (-1)
    #define BioAPI_UNSUPPORTED_BIR_HANDLE         (-2)
```

7.11 BioAPI_BIR_HEADER

Standard information which describes the content of the opaque biometric data (BDB) that follows. This information is readable by the application and is provided to allow it to make processing and routing decisions regarding the BIR. The header is not encrypted by the BSP.

```
typedef struct bioapi_bir_header {
       BioAPI_VERSION HeaderVersion;
       BioAPI_BIR_DATA_TYPE Type;
       BioAPI_BIR_BIOMETRIC_DATA_FORMAT Format;
       BioAPI_QUALITY Quality;
       BioAPI_BIR_PURPOSE Purpose;
       BioAPI_BIR_BIOMETRIC_TYPE FactorsMask;
       BioAPI_BIR_BIOMETRIC_PRODUCT_ID ProductID;
       BioAPI_DTG CreationDTG;
       BioAPI_BIR_SUBTYPE Subtype;
       BioAPI_DATE ExpirationDate;
       BioAPI_BIR_SECURITY_BLOCK_FORMAT SBFormat;
       BioAPI_UUID Index;
} BioAPI_BIR_HEADER;
```

NOTE 1: The BioAPI BIR Header corresponds to the SBH in CBEFF, ISO/IEC 19785-1.

NOTE 2: Expiration date corresponds to the 'Valid to' portion of the "CBEFF_BDB_validity_period" in ISO/IEC 19785-1. The Index field corresponds to the "CBEFF_BDB_index" in ISO/IEC 19785-1.

NOTE 3: It is possible that a BioAPI BIR may exist that has not been created by a BioAPI BSP but has been transformed from another data format. In this case, some of the header fields that are optional in CBEFF (ISO/IEC 19785-1) but required by BioAPI may not be present. For this reason, NO_VALUE_AVAILABLE or default values have been identified for these fields (within their corresponding data structures). However, all BIRs created by BioAPI BSPs shall include valid data for these fields and shall not use the NO VALUE AVAILABLE value. (The exceptions to this are BioAPI_QUALITY and BioAPI_BIR_SUBTYPE, which are optional in the BioAPI BIR header.) If such a non-BioAPI generated BIR is provided as an input parameter to a BioAPI BSP, the BSP may return an "Invalid BIR" error.

NOTE 4: The storage format for the BIR includes explicit length fields, which are not necessary in the C structure. See Annex B for the BIR storage format.

7.12 BioAPI_BIR_PURPOSE

7.12.1 A value which defines the purpose for which the BioAPI BIR is intended (when used as an input to a BioAPI function) or is suitable (when used as an output from a BioAPI function or within the BIR header).

```
typedef uint8_t BioAPI_BIR_PURPOSE;

    #define BioAPI_PURPOSE_VERIFY                              (1)
    #define BioAPI_PURPOSE_IDENTIFY                            (2)
    #define BioAPI_PURPOSE_ENROLL                              (3)
    #define BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY        (4)
    #define BioAPI_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY      (5)
    #define BioAPI_PURPOSE_AUDIT                               (6)
    #define BioAPI_NO_PURPOSE_AVAILABLE                        (0)
```

NOTE: The condition NO VALUE AVAILABLE is indicated by setting the value to zero. This value is used only for BIRs that are not originally generated by a BioAPI BSP, but originate from another source and have been transformed into a BioAPI BIR. BSPs shall not use this value.

7.12.2 The Purpose value is utilized in two ways. First, it is used as an input parameter to allow the application to indicate to the BSP the purpose for which the resulting BIR is intended, thus allowing the BSP to perform the appropriate capturing and/or processing to create the proper BIR for this purpose. The second use is within the BIR header to indicate to the application (or to the BSP during subsequent operations) what purpose the BIR is suitable for. For example, some BSPs use different BDB formats depending on whether the data is to be used for verification or identification, the latter generally including additional information to enhance speed or accuracy. Similarly, many BSPs use different data formats depending on whether the data is to be used as a sample for immediate verification or as a reference template for future matching (i.e., enrollment).

NOTE: The BioAPI BIR Purpose in a BIR header corresponds to the “CBEFF_BDB_purpose” in ISO/IEC 19785-1. The names differ slightly since the BioAPI BIR contains a single BDB, but the semantics are the same.

7.12.3 Restrictions on the use of BIR data of a particular purpose include:

a) All purposes are valid in the BIR header.

b) Purposes of BioAPI_PURPOSE_VERIFY and BioAPI_PURPOSE_IDENTIFY are only valid as input to the ***BioAPI_Capture*** function.

c) Purposes of BioAPI_PURPOSE_ENROLL, BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY, and BioAPI_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY are only valid as input to the ***BioAPI_Capture***, ***BioAPI_Enroll***, and ***BioAPI_Import*** functions.

d) The BioAPI_PURPOSE_AUDIT purpose is not valid as input to any function, but is only used in the BIR header.

e) The ***BioAPI_Process, BioAPI_CreateTemplate***, and ***BioAPI_ProcessWithAuxData*** functions do not have Purpose as an input parameter, but read the Purpose field from the BIR header of the input *CapturedBIR*.

f) The ***BioAPI_Process*** function may accept as input any intermediate BIR with a Purpose of BioAPI_PURPOSE_VERIFY or BioAPI_PURPOSE_IDENTIFY, and shall output only BIRs with the same purpose as the input BIR.

g) The ***BioAPI_CreateTemplate*** function may accept as input any intermediate BIR with a Purpose of BioAPI_PURPOSE_ENROLL, BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY, or BioAPI_PURPOSE_ENROLL_FOR_IDENTFICATION_ONLY, and shall output only BIRs with the same Purpose as the input BIR.

h) If a BIR is suitable for enrollment for either subsequent verification or subsequent identification, then the output BIR shall have a Purpose of BioAPI_PURPOSE_ENROLL.

7.13 BioAPI_BIR_SECURITY_BLOCK_FORMAT

Defines the format of the data contained within the security block (SB) of the BioAPI BIR (SecurityBlock element).

```
typedef struct bioapi_bir_security_block_format {
    uint16_t SecurityFormatOwner;
    uint16_t SecurityFormatType;
} BioAPI_BIR_SECURITY_BLOCK_FORMAT;
```

If neither the 'encrypt' or 'signed' flags are set within the BioAPI_BIR_DATA_TYPE field of the BIR Header, then the SecurityFormatOwner and SecurityFormatType are each set to 0x0000 and no security block is present.

SecurityFormatOwner values are assigned and registered by the CBEFF Registration Authority as Biometric Organization identifiers (see ISO/IEC 19785-2). Security Format Type is assigned by the Security Format Owner (the Biometric Organization) and may optionally be registered.

NOTE 1: Security Format IDs are analogous to Format IDs and the registration process is the same. A single vendor can register for one Format/Product/Security Owner value (a Biometric Organization identifier) that can be used in all associated fields.

NOTE 2: The content of the security block itself may include a digital signature or message authentication code (calculated on the BIR Header + BDB), BDB encryption parameters (e.g., encryption algorithm, key length), and/or BIR integrity parameters (e.g., algorithm ID, keyname, version).

NOTE 3: The BioAPI BIR security block Format in a BioAPI BIR header corresponds to the "CBEFF_SB_format_owner" and "CBEFF_SB_format_type" in ISO/IEC 19785-1.

7.14 BioAPI_BIR_SUBTYPE

7.14.1 This identifies a subtype within the BDB type (specified in the BioAPI_BIR_BIOMETRIC_TYPE). Its values and their meaning are defined by the specifier of that BDB type.

7.14.2 Each of the feature flags BioAPI_BIR_SUBTYPE_LEFT and BioAPI_BIR_SUBTYPE_RIGHT may or may not be set (zero, one, or both).

7.14.3 Either none or exactly one of the five finger subtypes may be set.

```
typedef uint8_t BioAPI_BIR_SUBTYPE;

    #define BioAPI_BIR_SUBTYPE_LEFT              (0x01)
    #define BioAPI_BIR_SUBTYPE_RIGHT             (0x02)
    #define BioAPI_BIR_SUBTYPE_THUMB             (0x04)
    #define BioAPI_BIR_SUBTYPE_POINTERFINGER     (0x08)
    #define BioAPI_BIR_SUBTYPE_MIDDLEFINGER      (0x10)
    #define BioAPI_BIR_SUBTYPE_RINGFINGER        (0x20)
    #define BioAPI_BIR_SUBTYPE_LITTLEFINGER      (0x40)
    #define BioAPI_BIR_SUBTYPE_MULTIPLE          (0x80)

    #define BioAPI_NO_SUBTYPE_AVAILABLE          (0x00)
```

NOTE 1: The condition NO VALUE AVAILABLE is indicated by setting the value to zero. BIRs that are not originally created by a BioAPI BSP but have been transformed from another data format and for which subtype information is not available may use this value.

NOTE 2: The BioAPI BIR Subtype corresponds to the "CBEFF_BDB_biometric_subtype" in ISO/IEC 19785-1.

NOTE 3: This structure is primarily used within a BIR header; however, it is also used as an input parameter for functions that capture biometric data. The BioAPI_NO_SUBTYPE_AVAILABLE value is used in the BIR header

when either this field is not applicable or information is not available. BioAPI_NO_SUBTYPE_AVAILABLE is also used as a function parameter when the application allows the BSP to determine which subtype is to be captured.

7.15 BioAPI_BOOL

This data type is used to indicate a true or false condition.

```
typedef  uint8_t BioAPI_BOOL;

    #define BioAPI_FALSE      (0)
    #define BioAPI_TRUE       (!BioAPI_FALSE)
```

7.16 BioAPI_BSP_SCHEMA

7.16.1 Information about a BSP, maintained in the component registry.

```
typedef struct bioapi_bsp_schema {
        BioAPI_UUID  BSPUuid;
        BioAPI_STRING  BSPDescription;
        uint8_t  *Path;
        BioAPI_VERSION  SpecVersion;
        BioAPI_STRING  ProductVersion;
        BioAPI_STRING  Vendor;
        BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *BSPSupportedFormats;
        uint32_t  NumSupportedFormats;
        BioAPI_BIR_BIOMETRIC_TYPE  FactorsMask;
        BioAPI_OPERATIONS_MASK  Operations;
        BioAPI_OPTIONS_MASK  Options;
        BioAPI_FMR  PayloadPolicy;
        uint32_t  MaxPayloadSize;
        int32_t  DefaultVerifyTimeout;
        int32_t  DefaultIdentifyTimeout;
        int32_t  DefaultCaptureTimeout;
        int32_t  DefaultEnrollTimeout;
        int32_t  DefaultCalibrateTimeout;
        uint32_t  MaxBSPDbSize;
        uint32_t  MaxIdentify;
    }BioAPI_BSP_SCHEMA;
```

7.16.2 Definitions

BSPUuid – UUID of the BSP.

BSPDescription – A NUL-terminated string containing a text description of the BSP.

Path – A pointer to a NUL-terminated string containing the path of the file containing the BSP executable code, including the filename. The path may be a URL. This string shall consist of ISO/IEC 10646 characters encoded in UTF-8 (see ISO/IEC 10646, Annex D).

NOTE: When BioAPI_BSP_SCHEMA is used within a function call, the component that receives the call allocates the memory for the *Path* schema element and the calling component frees the memory.

SpecVersion – Major/minor version number of the BioAPI specification to which the BSP was implemented.

ProductVersion – The version string of the BSP software.

Vendor – A NUL-terminated string containing the name of the BSP vendor.

BSPSupportedFormats – A pointer to an array of BioAPI_BIR_BIOMETRIC_DATA_FORMAT structures specifying the supported BDB formats.

NumSupportedFormats – Number of supported formats contained in BSPSupportedFormats.

FactorsMask – A mask which indicates which biometric types are supported by the BSP.

Operations – A mask which indicates which operations are supported by the BSP.

Options – A mask which indicates which options are supported by the BSP.

PayloadPolicy – Threshold setting (maximum FMR value) used to determine when to release the payload after successful verification.

MaxPayloadSize – Maximum payload size (in bytes) that the BSP can accept.

DefaultVerifyTimeout – Default timeout value in milliseconds used by the BSP for ***BioAPI_Verify*** operations when no timeout is specified by the application.

DefaultIdentifyTimeout – Default timeout value in milliseconds used by the BSP for ***BioAPI_Identify*** and ***BioAPI_IdentifyMatch*** operations when no timeout is specified by the application.

DefaultCaptureTimeout – Default timeout value in milliseconds used by the BSP for ***BioAPI_Capture*** operations when no timeout is specified by the application.

DefaultEnrollTimeout – Default timeout value in milliseconds used by the BSP for ***BioAPI_Enroll*** operations when no timeout is specified by the application.

DefaultCalibrateTimeout **–** Default timeout value in milliseconds used by the BSP for sensor calibration operations when no timeout is specified by the application.

MaxBSPDbSize – Maximum size of a BSP-controlled BIR database.

NOTE 1: Applies only when a BSP is only capable of directly managing a single archive unit.

NOTE 2: A value of zero means that no information about the database size is being provided for one of the following three reasons:

a) databases are not supported,

b) it is capable of managing multiple units (either directly or through a BFP interface), each of which may have a different “maximum size” and information about these units will be provided as part of the insert notification (part of Unit Schema), or

c) one archive unit is supported, but the information is not given here – it will be provided in the insert notification.

MaxIdentify – Largest population supported by the identify function. Unlimited = FFFFFFFF.

7.16.3 See clause 10.1.2 and 10.2.1 for further explanation of schema elements and for BSP insertion of information into the component registry.

7.17 BioAPI_CANDIDATE

One of a set of candidates returned by ***BioAPI_Identify*** or ***BioAPI_IdentifyMatch*** indicating a successful match.

```
typedef struct bioapi_candidate {
 BioAPI_IDENTIFY_POPULATION_TYPE Type;
 union {
     BioAPI_UUID *BIRInDataBase;
     uint32_t *BIRInArray;
 } BIR;
```

```
    BioAPI_FMR    FMRAchieved;
} BioAPI_CANDIDATE;
```

7.18 BioAPI_CATEGORY

This bitmask describes the category of BFP or BioAPI Unit. A BFP or BioAPI Unit shall belong to exactly one category.

```
typedef uint32_t BioAPI_CATEGORY;

#define BioAPI_CATEGORY_ARCHIVE          (0x00000001)
#define BioAPI_CATEGORY_MATCHING_ALG     (0x00000002)
#define BioAPI_CATEGORY_PROCESSING_ALG   (0x00000004)
#define BioAPI_CATEGORY_SENSOR           (0x00000008)
```

The highest significant bit is reserved and does not indicate a category of BFP or BioAPI Unit.

7.19 BioAPI_DATA

7.19.1 The BioAPI_DATA structure is used to associate a length, in bytes, with the address of an arbitrary block of contiguous memory.

```
typedef struct bioapi_data{
    uint32_t Length;  /* in bytes */
    void *Data;
} BioAPI_DATA;
```

7.19.2 Definitions

Length – Length of the data buffer in bytes.

Data – A pointer to the start of an arbitrary length data buffer.

7.20 BioAPI_DATE

Defines the date when the BIR was created or when it is no longer valid.

```
typedef struct bioapi_date {
    uint16_t Year; /* valid range: 1900 - 9999 */
    uint8_t Month; /* valid range: 01 - 12 */
    uint8_t Day;   /* valid range: 01 - 31, consistent with associated month/year */
} BioAPI_DATE;

    #define BioAPI_NO_YEAR_AVAILABLE     (0)
    #define BioAPI_NO_MONTH_AVAILABLE    (0)
    #define BioAPI_NO_DAY_AVAILABLE      (0)
```

The condition NO VALUE AVAILABLE is indicated by setting all fields to zero. When used within the Creation DTG within a BIR header, if no date information is available, then the NO_DATE_AVAILABLE value shall be used.

NOTE 1: The year 2000 AD is represented by a Year value of 2000.

NOTE 2: When used for ExpirationDate in a BIR header, corresponds to the “Valid to” portion of the “CBEFF_BDB_validity_period” in ISO/IEC 19785-1.

NOTE 3: Date formats are consistent with ISO 8601 [2].

7.21 BioAPI_DB_ACCESS_TYPE

This bitmask describes a biometric application's desired level of access to a BSP-controlled BIR database. The BSP may use the mask to determine what lock to obtain on the BIR database.

```
typedef uint32_t BioAPI_DB_ACCESS_TYPE;
    #define BioAPI_DB_ACCESS_READ        (0x00000001)
    #define BioAPI_DB_ACCESS_WRITE       (0x00000002)
```

7.22 BioAPI_DB_MARKER_HANDLE

A handle to a BIR database marker.

A marker is an internal (not standardized) data structure managed by a BSP, which dynamically points to a record in an open BSP-controlled BIR database. A marker handle is created and returned to a biometric application by the functions ***BioAPI_DbOpen*** and ***BioAPI_DbGetBIR***. The internal state of the marker includes the open database handle and also the position of a record in that open database. All markers (and their handles) held by a biometric application to an open BIR database become invalid when the database is closed by the biometric application.

```
typedef  uint32_t BioAPI_DB_MARKER_HANDLE;
```

7.23 BioAPI_DB_HANDLE

A handle to an open BIR database initially provided by a BSP to a biometric application and later used by that application to address the database through the BSP.

```
typedef  int32_t BioAPI_DB_HANDLE;

    #define BioAPI_DB_INVALID_HANDLE     (-1)
    #define BioAPI_DB_DEFAULT_HANDLE     (0)
    #define BioAPI_DB_DEFAULT_UUID_PTR   (NULL)
```

7.24 BioAPI_DBBIR_ID

A structure providing the handle to a BIR database controlled by a BSP and the UUID of a BIR in that database.

```
typedef struct bioapi_dbbir_id {
    BioAPI_DB_HANDLE DbHandle;
    BioAPI_UUID KeyValue;
} BioAPI_DBBIR_ID;
```

NOTE: This type is used as an element of BioAPI_INPUT_BIR.

7.25 BioAPI_DTG

Defines the date and time when the BIR was created.

```
typedef struct bioapi_DTG {
       BioAPI_DATE Date;
       BioAPI_TIME Time;
} BioAPI_DTG;
```

NOTE: The BioAPI DTG in a BIR header corresponds to the “CBEFF_creation_date” in ISO/IEC 19785-1.

7.26 BioAPI_EVENT

This enumeration defines the event types that can be raised by a BSP, BFP, or BioAPI Unit. Biometric applications can define event handling callback functions of type *BioAPI_EventHandler* to receive and manage these events. Callback functions are registered using the ***BioAPI_BSPLoad*** function. Example events include the addition (insertion) or removal of a biometric sensor. Events are asynchronous.

BioAPI_NOTIFY_SOURCE_PRESENT and BioAPI_NOTIFY_SOURCE_REMOVED events are generated by devices (sensor units) that can detect when the user may be available to provide a biometric sample (e.g., the user has placed a finger on a fingerprint device). BioAPI_NOTIFY_SOURCE_PRESENT indicates that a sample may be available, while BioAPI_NOTIFY_SOURCE_REMOVED indicates that a sample is probably no longer available. There is no requirement that these events must occur in pairs; several BioAPI_NOTIFY_SOURCE_PRESENT events may occur in succession.

```
typedef uint32_t BioAPI_EVENT;

    #define BioAPI_NOTIFY_INSERT          (1)
    #define BioAPI_NOTIFY_REMOVE          (2)
    #define BioAPI_NOTIFY_FAULT           (3)
    #define BioAPI_NOTIFY_SOURCE_PRESENT  (4)
    #define BioAPI_NOTIFY_SOURCE_REMOVED  (5)
```

7.27 BioAPI_EVENT_MASK

This type defines a mask with bit positions for each event type. The mask is used to enable/disable event notification, and to indicate what events are supported by a BSP.

```
typedef uint32_t BioAPI_EVENT_MASK;

    #define BioAPI_NOTIFY_INSERT_BIT          (0x00000001)
    #define BioAPI_NOTIFY_REMOVE_BIT          (0x00000002)
    #define BioAPI_NOTIFY_FAULT_BIT           (0x00000004)
    #define BioAPI_NOTIFY_SOURCE_PRESENT_BIT  (0x00000008)
    #define BioAPI_NOTIFY_SOURCE_REMOVED_BIT  (0x00000010)
```

NOTE: It may be impossible to mask an INSERT event coming from an attach session of a BSP, because the event may occur just after a ***BioAPI_BSPLoad*** call, before any ***BioAPI_EnableEvents*** call has had any chance to be processed. This is because ***BioAPI_EnableEvents*** requires a handle which is provided by ***BioAPI_BSPAttach***, and ***BioAPI_BSPAttach*** itself shall follow ***BioAPI_BSPLoad***. An INSERT event will be raised by the BSP on the ***BioAPI_BSPLoad*** call if a BioAPI Unit is already "inserted", and this event will reach the application before the application can call ***BioAPI_EnableEvents***.

7.28 BioAPI_EventHandler

7.28.1 This is the event handler interface that an application is required to support if it wishes to receive asynchronous notification of events such as the availability of a biometric sample or a fault detected by a BSP. The event handler function is registered with the BioAPI Framework as part of the ***BioAPI_BSPLoad*** function. This is the caller's event handler for all general BSP events over all of the caller's attach sessions with the loaded BSP. General event notifications are processed through the BioAPI Framework.

The event handler function and any functions invoked (directly or indirectly) by that function shall not issue BioAPI calls. These circular calls may result in deadlock in numerous situations; hence the event handler shall be implemented without using BioAPI services (however, enumeration functions can be used at any time).

The BioAPI_EventHandler can be invoked multiple times in response to a single event occurring in the associated BSP. The event handler and the calling application shall track receipt of event notifications and ignore duplicates. An event notification provides the following information:

```
typedef BioAPI_RETURN (BioAPI *BioAPI_EventHandler)
       (const BioAPI_UUID  *BSPUuid,
       BioAPI_UNIT_ID  UnitID,
       void*  AppNotifyCallbackCtx,
       const BioAPI_UNIT_SCHEMA  *UnitSchema,
       BioAPI_EVENT  EventType);
```

7.28.2 Definitions

BSPUuid – The UUID of the BSP raising the event.

UnitID – The unit ID of the BioAPI Unit associated with the event.

AppNotifyCallbackCtx – A generic pointer to context information that was provided in the call to ***BioAPI_BSPLoad*** that established the event handler.

UnitSchema – A pointer to the unit schema of the BioAPI Unit associated with the event.

EventType – The BioAPI_EVENT that has occurred.

If the *EventType* is BioAPI_NOTIFY_INSERT, then a unit schema shall be provided (that is, *UnitSchema* shall point to a variable of type BioAPI_UNIT_SCHEMA). Otherwise, *UnitSchema* shall be NULL.

When the application receives a call to an event handler that carries a unit schema, the application shall not call ***BioAPI_Free*** to free the memory block containing the unit schema or a memory block pointed to by any of its members.

7.29 BioAPI_FMR

A 32-bit integer value (N) that indicates a probable False Match Rate of $N/(2^{31}-1)$. The larger the value, the worse the result. Negative values are used to signal exceptional conditions. The only negative value currently defined is minus one.

```
typedef   int32_t   BioAPI_FMR;

    #define BioAPI_NOT_SET          (-1)
```

NOTE: FMR is used within BioAPI as a means of setting thresholds and returning scores (see clause C.4).

7.30 BioAPI_FRAMEWORK_SCHEMA

7.30.1 The framework schema entry for a BioAPI Framework as recorded in the component registry.

```
typedef struct bioapi_framework_schema {
        BioAPI_UUID  FrameworkUuid;
        BioAPI_STRING  FwDescription;
        uint8_t  *Path;
        BioAPI_VERSION  SpecVersion;
        BioAPI_STRING  ProductVersion;
        BioAPI_STRING  Vendor;
        BioAPI_UUID  FwPropertyId;
        BioAPI_DATA FwProperty;
    } BioAPI_FRAMEWORK_SCHEMA;
```

7.30.2 Definitions

FrameworkUuid – UUID of the Framework component.

FwDescription – A NUL-terminated string containing a text description of the Framework.

Path – A pointer to a NUL-terminated string containing the path of the file containing the Framework executable code, including the filename. The path may be a URL. This string shall consist of ISO/IEC 10646 characters encoded in UTF-8 (see ISO/IEC 10646, Annex D).

NOTE: When BioAPI_FRAMEWORK_SCHEMA is used within a function call, the component that receives the call allocates the memory for the *Path* schema element and the calling component frees the memory.

SpecVersion – Major/minor version number of the BioAPI specification to which the Framework was implemented.

ProductVersion – The version string of the Framework software.

Vendor – A NUL-terminated string containing the name of the Framework vendor.

FwPropertyID – UUID of the format of the following Framework property.

FwProperty – Address and length of a memory buffer containing the Framework property. The format and content of the Framework property can either be specified by a vendor or can be specified in a related standard.

7.30.3 See clause 10.1.1 for further explanation of framework schema elements.

7.31 BioAPI_GUI_BITMAP

7.31.1 This structure provides a graphic for display by the application.

```
typedef struct bioapi_gui_bitmap {
    uint32_t  Width;   /* Width of bitmap in pixels (number of pixels for each line) */
    uint32_t  Height;  /* Height of bitmap in pixels (number of lines) */
    BioAPI_DATA  Bitmap;
} BioAPI_GUI_BITMAP;
```

7.31.2 Definitions

Bitmap - a series of 8-bit grayscale pixels (where '00' = black and 'FF' = white), read from left to right, top to bottom, as follows:

Table 1 — GUI Bitmap Format

Byte Position	Meaning	Description
0	line 0, pixel 0	first pixel of first line
1	line 0, pixel 1	second pixel of first line
…	…	
width-1	line 0, pixel(width-1)	last pixel of first line
width	line 1, pixel 0	first pixel of second line
…	…	
(width * height) - 1	line (width – 1), pixel (height – 1)	last line, last pixel

NOTE: This is used with the application-controlled GUI option. See BioAPI_GUI_STATE_CALLBACK and BioAPI_GUI_STREAMING_CALLBACK descriptions under 7.36 and 7.37.

7.32 BioAPI_GUI_MESSAGE

This structure provides a message for display by the application.

```
typedef uint32_t BioAPI_GUI_MESSAGE;
```

NOTE: This is used with the application-controlled GUI option. See BioAPI_GUI_STATE_CALLBACK description under 7.36.

7.33 BioAPI_GUI_PROGRESS

A value that indicates the percentage progress (% complete) for an in-progress BSP operation to the application for display purposes.

```
typedef uint8_t BioAPI_GUI_PROGRESS;
```

NOTE: This is used with the application-controlled GUI option. See BioAPI_GUI_STATE_CALLBACK description under 7.36.

7.34 BioAPI_GUI_RESPONSE

Return value from the biometric application during a callback operation.

```
typedef uint8_t BioAPI_GUI_RESPONSE;

    #define BioAPI_CAPTURE_SAMPLE      (1)     /* Instruction to BSP to capture sample */
    #define BioAPI_CANCEL              (2)     /* User cancelled operation */
    #define BioAPI_CONTINUE            (3)     /* User or application selects to proceed */
    #define BioAPI_VALID_SAMPLE        (4)     /* Valid sample received */
    #define BioAPI_INVALID_SAMPLE      (5)     /* Invalid sample received */
```

NOTE: This is used with the application controlled GUI option. See BioAPI_GUI_STATE_CALLBACK description under 7.36.

7.35 BioAPI_GUI_STATE

A mask that indicates GUI state, and also what other parameter values are provided in the GUI State Callback (this information is provided by the BSP through the GUI State Callback to the application).

```
typedef uint32_t BioAPI_GUI_STATE;

    #define BioAPI_SAMPLE_AVAILABLE  (0x0001)   /* Sample captured and available */
    #define BioAPI_MESSAGE_PROVIDED  (0x0002)   /* BSP provided message for display */
    #define BioAPI_PROGRESS_PROVIDED (0x0004)   /* BSP provide progress for display */
```

NOTE: This is used with the application-controlled GUI option. See BioAPI_GUI_STATE_CALLBACK description under 7.36.

7.36 BioAPI_GUI_STATE_CALLBACK

7.36.1 This structure is a function pointer type – a callback function that an application supplies to allow the biometric service provider to pass GUI state information to the application through the Framework, and to receive responses back.

7.36.2 Function

```
typedef BioAPI_RETURN (BioAPI *BioAPI_GUI_STATE_CALLBACK)
      (void  *GuiStateCallbackCtx,
       BioAPI_GUI_STATE  GuiState,
       BioAPI_GUI_RESPONSE  *Response,
       BioAPI_GUI_MESSAGE  Message,
       BioAPI_GUI_PROGRESS  Progress,
       const BioAPI_GUI_BITMAP  *SampleBuffer);
```

Return of a value different from BioAPI_OK will make the calling function (e.g., ***BioAPI_Enroll***) to immediately return to the caller with that value as its error code.

7.36.3 Parameters

GuiStateCallbackCtx (input) - A generic pointer to context information that was provided by the original requester and is being returned to its originator.

GuiState (input) – an indication of the current state of the BSP with respect to the GUI, plus an indication of what others parameters are available.

Response (output) – A pointer to the response from the application back to the biometric service provider on return from the Callback.

Message (input/optional) – The number of a message to display to the user. Message numbers and message text are not standardized, and are BSP dependent. *GuiState* indicates if a *Message* is provided; if not, the parameter is NULL.

Progress (input/optional) – A value that indicates (as a percentage) the amount of progress in the development of a Sample/BIR. The value may be used to display a progress bar. Not all BSPs support a progress indication. *GuiState* indicates if a sample *Progress* value is provided in the call; if not, the parameter is NULL.

SampleBuffer (input/optional) – The current sample buffer for the application to display. *GuiState* indicates if a sample *Buffer* is provided; if not, the parameter is NULL.

NOTE: See also clause C.7 on User Interface Considerations.

7.37 BioAPI_GUI_STREAMING_CALLBACK

7.37.1 This is a function pointer type – a callback function that a biometric application supplies to allow a BSP to stream data for display, in the form of a sequence of bitmaps, to the application through the Framework.

7.37.2 Function

```
typedef BioAPI_RETURN (BioAPI *BioAPI_GUI_STREAMING_CALLBACK)
       (void *GuiStreamingCallbackCtx,
       const BioAPI_GUI_BITMAP *Bitmap);
```

Return of a value different from BioAPI_OK will make the calling function (e.g., ***BioAPI_Enroll***) to immediately return to the caller with that value as its error code.

7.37.3 Parameters

GuiStreamingCallbackCtx (input) – A generic pointer to context information that was provided by the original requester and is being returned to its originator.

Bitmap (input) – a pointer to the bitmap to be displayed.

NOTE: See also clause C.7 on User Interface Considerations.

7.38 BioAPI_HANDLE

A unique identifier, returned on ***BioAPI_BSPAttach***, that identifies an attached BioAPI BSP session.

```
typedef uint32_t BioAPI_HANDLE;
```

7.39 BioAPI_IDENTIFY_POPULATION

A structure used to identify the set of BIRs to be used as input to a ***BioAPI_Identify*** or ***BioAPI_IdentifyMatch*** operation.

```
typedef struct bioapi_identify_population {
    BioAPI_IDENTIFY_POPULATION_TYPE  Type;
    union {
        BioAPI_DB_HANDLE  *BIRDataBase;
        BioAPI_BIR_ARRAY_POPULATION  *BIRArray;
    } BIRs;
} BioAPI_IDENTIFY_POPULATION;
```

If BioAPI_PRESET_ARRAY_TYPE is specified in Type, BIRArray shall be selected and shall be set to NULL.

7.40 BioAPI_IDENTIFY_POPULATION_TYPE

A value indicating the method of BIR input to a ***BioAPI_Identify*** or ***BioAPI_IdentifyMatch*** operation, whether it is via a passed-in array or a pointer to a BIR database.

```
typedef uint8_t BioAPI_IDENTIFY_POPULATION_TYPE;

    #define   BioAPI_DB_TYPE             (1)
    #define   BioAPI_ARRAY_TYPE          (2)
    #define   BioAPI_PRESET_ARRAY_TYPE   (3)
```

7.41 BioAPI_INDICATOR_STATUS

This type is used by a biometric application to get and set device (BioAPI Unit) indicators supported by a BSP. The precise physical form of these indicators (and their availability) is implementation-dependent.

```
typedef uint8_t BioAPI_INDICATOR_STATUS;

    #define   BioAPI_INDICATOR_ACCEPT    (1)
    #define   BioAPI_INDICATOR_REJECT    (2)
    #define   BioAPI_INDICATOR_READY     (3)
    #define   BioAPI_INDICATOR_BUSY      (4)
    #define   BioAPI_INDICATOR_FAILURE   (5)
```

7.42 BioAPI_INPUT_BIR

A structure used to input a BioAPI BIR to the API. Such input can be in one of three forms:

a) a BIR Handle;

b) a key to a BIR in a database controlled by the BSP. If the *DbHandle* is zero, a database selected by the BSP is assumed. (A *DbHandle* is returned when a BIR database is opened);

c) a BioAPI BIR structure.

```
typedef struct bioapi_input_bir {
    BioAPI_INPUT_BIR_FORM  Form;
    union {
        BioAPI_DBBIR_ID  *BIRinDb;
        BioAPI_BIR_HANDLE  *BIRinBSP;
        BioAPI_BIR  *BIR;
    } InputBIR;
} BioAPI_INPUT_BIR;
```

7.43 BioAPI_INPUT_BIR_FORM

A value indicating the form/method by which a BIR is provided.

```
typedef uint8_t BioAPI_INPUT_BIR_FORM;

    #define     BioAPI_DATABASE_ID_INPUT      (1)
    #define     BioAPI_BIR_HANDLE_INPUT       (2)
    #define     BioAPI_FULLBIR_INPUT          (3)
```

NOTE: This type is used as an element of BioAPI_INPUT_BIR.

7.44 BioAPI_INSTALL_ACTION

Specifies the action to be taken during BSP installation operations.

```
typedef uint32_t  BioAPI_INSTALL_ACTION;

    #define     BioAPI_INSTALL_ACTION_INSTALL        (1)
    #define     BioAPI_INSTALL_ACTION_REFRESH        (2)
    #define     BioAPI_INSTALL_ACTION_UNINSTALL      (3)
```

7.45 BioAPI_INSTALL_ERROR

A structure which contains additional information regarding an error state that has occurred during an installation operation.

```
typedef struct install_error{
    BioAPI_RETURN ErrorCode;
    BioAPI_STRING ErrorString;
} BioAPI_INSTALL_ERROR;
```

7.46 BioAPI_OPERATIONS_MASK

A mask that indicates what operations are supported by the biometric service provider.

```
typedef uint32_t BioAPI_OPERATIONS_MASK;

        #define BioAPI_ENABLEEVENTS                 (0x00000001)
        #define BioAPI_SETGUICALLBACKS              (0x00000002)
        #define BioAPI_CAPTURE                      (0x00000004)
        #define BioAPI_CREATETEMPLATE               (0x00000008)
        #define BioAPI_PROCESS                      (0x00000010)
        #define BioAPI_PROCESSWITHAUXBIR            (0x00000020)
        #define BioAPI_VERIFYMATCH                  (0x00000040)
        #define BioAPI_IDENTIFYMATCH                (0x00000080)
        #define BioAPI_ENROLL                       (0x00000100)
        #define BioAPI_VERIFY                       (0x00000200)
        #define BioAPI_IDENTIFY                     (0x00000400)
        #define BioAPI_IMPORT                       (0x00000800)
        #define BioAPI_PRESETIDENTIFYPOPULATION     (0x00001000)
        #define BioAPI_DATABASEOPERATIONS           (0x00002000)
        #define BioAPI_SETPOWERMODE                 (0x00004000)
        #define BioAPI_SETINDICATORSTATUS           (0x00008000)
        #define BioAPI_GETINDICATORSTATUS           (0x00010000)
        #define BioAPI_CALIBRATESENSOR              (0x00020000)
        #define BioAPI_UTILITIES                    (0X00040000)
        #define BioAPI_QUERYUNITS                   (0x00100000)
        #define BioAPI_QUERYBFPS                    (0x00200000)
        #define BioAPI_CONTROLUNIT                  (0X00400000)
```

NOTE: This type is used as an element of BioAPI_BSP_SCHEMA.

7.47 BioAPI_OPTIONS_MASK

A mask that indicates what options are supported by the biometric service provider. Note that optional functions are identified within the BioAPI_OPERATIONS_MASK and not repeated here.

```
typedef uint32_t BioAPI_OPTIONS_MASK;

#define    BioAPI_RAW                              (0x00000001)
```
If set, indicates that the BSP supports the return of raw/audit data.

```
#define    BioAPI_QUALITY_RAW                      (0x00000002)
```
If set, the BSP supports the return of a quality value (in the BIR header) for raw biometric data.

```
#define    BioAPI_QUALITY_INTERMEDIATE             (0x00000004)
```
If set, the BSP supports the return of a quality value (in the BIR header) for intermediate biometric data.

```
#define    BioAPI_QUALITY_PROCESSED                (0x00000008)
```
If set, the BSP supports the return of quality value (in the BIR header) for processed biometric data.

```
#define    BioAPI_APP_GUI                          (0x00000010)
```
If set, the BSP supports application control of the GUI.

```
#define    BioAPI_STREAMINGDATA                    (0x00000020)
```
If set, the BSP provides GUI streaming data.

```
#define    BioAPI_SOURCEPRESENT                    (0x00000040)
```
If set, the BSP supports the detection of the presence a biometric characteristic at the biometric sensor.

```
#define    BioAPI_PAYLOAD                          (0x00000080)
```
If set, the BSP supports payload carry (accepts payload during ***BioAPI_Enroll or BioAPI_CreateTemplate*** and returns payroll upon successful ***BioAPI_Verify or BioAPI_VerifyMatch***).

```
#define    BioAPI_BIR_SIGN                         (0x00000100)
```
If set, the BSP returns signed BIRs.

```
#define    BioAPI_BIR_ENCRYPT                      (0x00000200)
```
If set, the BSP returns encrypted BIRs (BDB portion).

```
#define    BioAPI_TEMPLATEUPDATE                   (0x00000400)
```
If set, the BSP updates any provided input reference template as part of the enrollment or template creation operation.

```
#define    BioAPI_ADAPTATION                       (0x00000800)
```
If set, the BSP supports BIR adaptation in the return parameters of a ***Verify*** or ***VerifyMatch*** operation.

```
#define    BioAPI_BINNING                          (0x00001000)
```
If set, the BSP supports binning (in ***BioAPI_Identify*** and ***BioAPI_IdentifyMatch*** operations).

```
#define    BioAPI_SELFCONTAINEDDEVICE              (0x00002000)
```
If set, the BSP supports a self-contained device.

```
#define    BioAPI_MOC                              (0x00004000)
```
If set, the BSP directly supports matching on a smartcard.

```
#define    BioAPI_SUBTYPE_TO_CAPTURE               (0x00008000)
```
If set, the BSP supports specification by the application of which biometric subtype to capture and will act on it.

```
#define    BioAPI_SENSORBFP                        (0x00010000)
```
If set, the BSP supports management of a sensor unit through a BFP.

```
#define    BioAPI_ARCHIVEBFP                    (0x00020000)
```
If set, the BSP supports management of an archive unit through a BFP.

```
#define    BioAPI_MATCHINGBFP                   (0x00040000)
```
If set, the BSP supports management of a matching algorithm unit through a BFP.

```
#define    BioAPI_PROCESSINGBFP                   (0x00080000)
```
If set, the BSP supports managment of a processing algorithm unit through a BFP.

```
#define    BioAPI_COARSESCORES                  (0x00100000)
```
If set, the BSP returns scores which are coarsely quantized (see clause C.4.6).

NOTE: This type is used as an element of BioAPI_BSP_SCHEMA.

7.48 BioAPI_POWER_MODE

An enumeration that specifies the types of power modes the BSP and its attached devices (BioAPI Units) will try to use.

```
typedef uint32_t BioAPI_POWER_MODE;

    /* All functions available */
    #define BioAPI_POWER_NORMAL      (1)

    /* Able to detect (for example) insertion/finger on/person present type of events */
    #define BioAPI_POWER_DETECT      (2)

    /* Minimum mode. All functions off */
    #define BioAPI_POWER_SLEEP       (3)
```

7.49 BioAPI_QUALITY

7.49.1 A value indicating the relative quality of the biometric data in the BDB.

```
typedef  int8_t  BioAPI_QUALITY;
```

7.49.2 The performance of biometrics varies with the quality of the biometric sample. Since a universally accepted definition of quality does not exist, BioAPI specifies the following structure with the goal of relatively quantifying the effect of quality on usage of the BIR (as envisioned by the BSP implementer). The scores as reported by the BSP are based on the purpose (BioAPI_BIR_PURPOSE) indicated by the application (e.g., BioAPI_PURPOSE_ENROLL, BioAPI_PURPOSE_VERIFY, BioAPI_PURPOSE_IDENTIFY, etc.). Additionally, the demands upon the biometric vary based on the actual customer application and/or environment (i.e., a particular application usage may require higher quality samples than would normally be required by less demanding applications).

7.49.3 Quality measurements are reported as an integral value in the range 0-100 except as follows:

Value of "-1": BioAPI_QUALITY was not set by the BSP (reference the BSP implementer's documentation for an explanation).

Value of "-2": BioAPI_QUALITY is not supported by the BSP.

7.49.4 There are two objectives in providing BioAPI_QUALITY feedback to the biometric application:

c) The primary objective is to have the BSP inform the application how suitable the biometric sample is for the purpose (BioAPI_PURPOSE) specified by the application (as intended by the BSP implementer based on the use scenario envisioned by that BSP implementer).

d) The secondary objective is to provide the application with relative results (e.g., current sample is better/worse than previous sample).

7.49.5 Quality scores in the range 0-100 have the following general interpretation:

0-25: UNACCEPTABLE: The BDB cannot be used for the purpose specified by the application (BioAPI_PURPOSE). The BDB needs to be replaced using one or more new biometric samples.

26-50: MARGINAL: The BDB will provide poor performance for the purpose specified by the application (BioAPI_PURPOSE) and in most application environments will compromise the intent of the application. The BDB needs to be replaced using one or more new biometric samples.

51-75: ADEQUATE: The biometric data will provide good performance in most application environments based on the purpose specified by the application (BioAPI_PURPOSE). The application should attempt to obtain higher quality data if the application developer anticipates demanding usage.

76-100: EXCELLENT: The biometric data will provide good performance for the purpose specified by the application (BioAPI_BIR_PURPOSE).

NOTE: The BioAPI Quality corresponds to the "CBEFF_BDB_quality" in ISO/IEC 19785-1.

7.50 BioAPI_RETURN

7.50.1 This data type is returned by all BioAPI functions. The permitted values include:

- BioAPI_OK
- All Error Values defined in this specification
- BSP-specific error values defined and returned by a specific biometric service provider
- All Error Values defined for lower level components
- BioAPI Unit-specific error values defined and returned by a specific biometric service provider

```
typedef uint32_t BioAPI_RETURN;

    #define BioAPI_OK  (0)
```

7.50.2 Definitions

BioAPI_OK – Indicates operation was successful

All other values – Indicates the operation was unsuccessful and identifies the specific, detected error that resulted in the failure. (See clause 11 for the list of standardised error codes.)

7.51 BioAPI_STRING

7.51.1 This is used by BioAPI data structures to represent a character string inside of a fixed-length buffer. The character string shall be NUL-terminated.

```
typedef uint8_t BioAPI_STRING [269];
```

7.51.2 This string shall consist of ISO/IEC 10646 characters encoded in UTF-8 (see ISO/IEC 10646, Annex D).

7.52 BioAPI_TIME

Defines the time when the BIR was created.

```
typedef struct bioapi_time {
    uint8_t Hour;      /* valid range: 00 – 23, 99 */
    uint8_t Minute ;   /* valid range: 00 – 59, 99 */
    uint8_t Second ;   /* valid range: 00 – 59, 99 */
} BioAPI_TIME;

    #define BioAPI_NO_HOUR_AVAILABLE         (99)
    #define BioAPI_NO_MINUTE_AVAILABLE       (99)
    #define BioAPI_NO_SECOND_AVAILABLE       (99)
```

The condition NO VALUE AVAILABLE shall be indicated by setting all fields to the value 99 (decimal). When used within the Creation DTG within a BIR header, if no time information is available, then the BioAPI_NO_HOUR_AVAILABLE, BioAPI_NO_MINUTE_AVAILABLE, and BioAPI_NO_SECOND_AVAILABLE values shall be used.

NOTE: Time formats are consistent with ISO 8601 [2].

7.53 BioAPI_UNIT_ID

A BioAPI Unit ID is a 32-bit integer assigned to a BioAPI Unit by a BSP that manages (directly or indirectly) the BioAPI Unit.

```
typedef  uint32_t  BioAPI_UNIT_ID;

#define BioAPI_DONT_CARE      (0x00000000)
#define BioAPI_DONT_INCLUDE   (0xFFFFFFFF)
```

7.54 BioAPI_UNIT_LIST_ELEMENT

7.54.1 Indicates a BioAPI Unit by category and ID. A list of these elements is used to establish an attach session.

```
typedef struct bioapi_unit_list_element  {
        BioAPI_CATEGORY  UnitCategory;
        BioAPI_UNIT_ID  UnitId;
    } BioAPI_UNIT_LIST_ELEMENT;
```

7.54.2 Definitions

UnitCategory – Defines the category of the BioAPI Unit.

UnitId – The ID of a BioAPI Unit to be used in this attach session. This will be created by the BSP uniquely.

7.55 BioAPI_UNIT_SCHEMA

7.55.1 A BioAPI Unit schema indicates the specific characteristics of the BioAPI Unit.

```
typedef struct bioapi_unit_schema {
        BioAPI_UUID  BSPUuid;
        BioAPI_UUID  UnitManagerUuid;
        BioAPI_UNIT_ID  UnitId;
        BioAPI_CATEGORY  UnitCategory;
        BioAPI_UUID  UnitProperties;
        BioAPI_STRING  VendorInformation;
        uint32_t  SupportedEvents;
        BioAPI_UUID  UnitPropertyID;
```

```
    BioAPI_DATA  UnitProperty;
    BioAPI_STRING  HardwareVersion;
    BioAPI_STRING  FirmwareVersion;
    BioAPI_STRING  SoftwareVersion;
    BioAPI_STRING  HardwareSerialNumber;
    BioAPI_BOOL  AuthenticatedHardware;
    uint32_t  MaxBSPDbSize;
    uint32_t  MaxIdentify;
} BioAPI_UNIT_SCHEMA;
```

NOTE: The BioAPI Unit Schema is used as a function parameter (by ***BioAPI_QueryUnits*** and *BioAPI_EventHandler*), but is not stored in the component registry.

7.55.2 Definitions

BSPUuid – UUID of the BSP supporting this BioAPI Unit.

UnitManagerUuid – UUID of the software component directly managing the BioAPI Unit (may be either the BSP itself or a BFP).

UnitId – ID of the BioAPI Unit during this attach session. This will be created by the BSP uniquely.

UnitCategory – Defines the category of the BioAPI Unit.

UnitProperties – UUID indicating a set of properties of the BioAPI Unit. The indicated set can either be specified by each vendor or follow a related standard.

VendorInformation – Contains vendor proprietary information.

SupportedEvents – A mask indicating which types of events are supported by the hardware.

UnitPropertyID – UUID of the format of the following Unit property structure.

UnitProperty – Address and length of a memory buffer containing the Unit property describing the BioAPI Unit. The format and content of the Unit property can either be specified by the vendor or be specified in a related standard.

HardwareVersion – A NUL-terminated string containing the version of the hardware. Empty if not available.

FirmwareVersion – A NUL-terminated string containing the version of the firmware. Empty if not available.

SoftwareVersion – A NUL-terminated string containing the version of the software. Empty if not available.

HardwareSerialNumber – A NUL-terminated string containing the vendor defined unique serial number of the hardware component. Empty if not available.

AuthenticatedHardware – A boolean value that indicates whether the hardware component has been authenticated.

MaxBSPDbSize – Maximum size database supported by the BioAPI Unit. If zero, no database exists.

MaxIdentify – Largest identify population supported by the BioAPI Unit. Unlimited = FFFFFFFF.

7.56 BioAPI_UUID

A universally unique identifier used to identify and address components (BSPs, BFPs, BioAPI Units, and the BioAPI Framework), to identify BIR databases, and as a database index for BSP-controlled BIR databases.

```
typedef  uint8_t  BioAPI_UUID[16];
```

NOTE 1: UUIDs are defined in ISO/IEC 9834-8.

NOTE 2: The BioAPI UUID within a BIR header corresponds to the “CBEFF_BDB_index” in ISO/IEC 19785-1.

7.57 BioAPI_VERSION

7.57.1 This type is used to represent the version of the BioAPI or FPI specification to which components or data have been implemented. The primary use of this value is within the BIR header and within schemas of the component registry.

The version corresponding to this part of ISO/IEC 19784 has an integer value of (decimal) 32, or (hex) 20, corresponding to a Major value of 2 and a Minor value of zero.

```
typedef  uint8_t  BioAPI_VERSION;
```

NOTE 1: This type is not used for product versions, which are generally represented by strings.

NOTE 2: The BioAPI Version used within the BIR header corresponds to the “CBEFF_patron_header_version” in ISO/IEC 19785-1.

7.57.2 Definitions

The BioAPI Version is a concatenation of the major and minor version values such that first hex digit represents the major version and the second hex digit represents the minor version:

```
BioAPI_VERSION      0xnm
```

where n = MajorVersion and m=MinorVersion.

8 BioAPI functions

8.1 Component Management Functions

The following functions are handled by the BioAPI Framework. Some are handled internally and others are passed through to a BSP via the SPI.

8.1.1 BioAPI_Init

```
BioAPI_RETURN BioAPI BioAPI_Init
    (BioAPI_VERSION  Version);
```

8.1.1.1 Description

This function initializes the BioAPI Framework and verifies that the version of the BioAPI Framework expected by the application is compatible with the version of the BioAPI Framework on the system. This function should be called at least once by the application.

Any call to ***BioAPI_Init*** which occurs while there are previous calls to ***BioAPI_Init*** that have not been followed by a corresponding call to ***BioAPI_Terminate*** will be handled as follows: The BioAPI Framework will respond with either BioAPI_OK (if and only if the handling of the proposed version number by the framework is compatible with the handling of the version number that was proposed by the previous ***BioAPI_Init*** call) or BioAPIERR_INCOMPATIBLE_VERSION, but will not reinitialize. A count of the number of successful ***BioAPI_Init*** calls will be maintained by the Framework, which will not terminate until a corresponding number of ***BioAPI Terminate*** calls have been issued.

This function is handled internally within the BioAPI Framework and is not passed through to a BSP.

8.1.1.2 Parameters

Version (input) – the major and minor version number of the BioAPI specification that the biometric application is compatible with.

8.1.1.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.1.1.4 Errors

`BioAPIERR_INCOMPATIBLE_VERSION`

See also **BioAPI Error Handling** (clause 11).

8.1.2 BioAPI_Terminate

```
BioAPI_RETURN BioAPI BioAPI_Terminate (void);
```

8.1.2.1 Description

This function terminates a biometric application's use of the BioAPI Framework. The Framework can cleanup all internal state associated with the calling application.

This function shall only be called if there is at least one successful call to ***BioAPI_Init*** for which a corresponding call to this function has not yet been made. ***BioAPI_Terminate*** shall not perform any actions if there are any outstanding calls to ***BioAPI_Init*** (i.e., calls to ***BioAPI_Init*** that have not been followed by corresponding calls to ***BioAPI_Terminate***).

This function should not be called by the application if there is a call to ***BioAPI_BSPLoad*** for which a corresponding call to ***BioAPI_BSPUnload*** (for a given BSP UUID) has not yet been made. However, should this function be called while there are still BSPs loaded, then for each call to ***BioAPI_BSPLoad*** without a corresponding call to ***BioAPI_BSPUnload***, the actions relative to the missing corresponding call to ***BioAPI_BSPUnload*** shall be implicitly performed by the Framework (as though the corresponding function had been called at that time), followed by the actions relative to ***BioAPI_Terminate*** (i.e, the Framework shall unload any BSPs prior to terminating).

This function is handled internally within the BioAPI Framework and is not passed through to a BSP, except where a ***BioAPI_BSPUnload*** is implied, as specified above.

8.1.2.2 Parameters

None.

8.1.2.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

NOTE: A ***BioAPI_Terminate*** operation is typically not expected to fail; however, should an anomaly condition occur, the application is not required to take any further action.

8.1.2.4 Errors

See **BioAPI Error Handling** (clause 11).

8.1.3 BioAPI_GetFrameworkInfo

```
BioAPI_RETURN BioAPI BioAPI_GetFrameworkInfo
    (BioAPI_FRAMEWORK_SCHEMA  *FrameworkSchema);
```

8.1.3.1 Description

This function returns information about the BioAPI Framework itself. Since multiple frameworks may exist on a computer, applications will need information about them in order to choose the one to use.

8.1.3.2 Parameters

FrameworkSchema (output) – A pointer to memory where the schema information will be returned.

8.1.3.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.1.3.4 Errors

See **BioAPI Error Handling** (clause 11).

8.1.4 BioAPI_EnumBSPs

```
BioAPI_RETURN BioAPI BioAPI_EnumBSPs
    (BioAPI_BSP_SCHEMA  **BSPSchemaArray,
    uint32_t  *NumberOfElements);
```

8.1.4.1 Description

This function provides information about all BSPs currently installed in the component registry. It performs the following actions (in order):

a) allocates a memory block large enough to contain an array of elements of type BioAPI_BSP_SCHEMA with as many elements as the number of installed BSPs; .

b) fills the array with the BSP schemas of all installed BSPs; and

c) returns the address of the array in the *BSPSchemaArray* parameter and the number of elements of the array in the *NumberOfElements* parameter.

This function shall only be called if there is at least one call to ***BioAPI_Init*** for which a corresponding call to ***BioAPI_Terminate*** has not yet been made.

This function is handled internally within the BioAPI Framework and is not passed through to any BSP.

The memory block containing the array shall be freed by the application via a call to ***BioAPI_Free*** (see clause 8.7.2) when it is no longer needed by the application. The memory block pointed to by the *Path* member within each element of the array shall also be freed by the application via a call to ***BioAPI_Free*** when it is no longer needed by the application.

8.1.4.2 Parameters

BSPSchemaArray (output) – A pointer to the address of the array of elements of the type BioAPI_BSP_SCHEMA (allocated by the framework) containing the BSP schema information.

NumberOfElements (output) – A pointer to the number of elements of the array (which is also the number of BSP schemas in the component registry).

8.1.4.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.1.4.4 Errors

See **BioAPI Error Handling** (clause 11).

8.1.5 BioAPI_BSPLoad

```
BioAPI_RETURN BioAPI BioAPI_BSPLoad
    (const BioAPI_UUID  *BSPUuid,
    BioAPI_EventHandler  AppNotifyCallback,
    void*  AppNotifyCallbackCtx);
```

8.1.5.1 Description

This function initializes a BSP using the ***BioSPI_BSPLoad*** (see clause 9.3.1.1). Initialization includes registering the application's event handler for the specified BSP and enabling all events. The application can choose to provide an event handler function to receive notification of events. Many applications can independently and concurrently load the same BSP, and each application can establish its own event handler. They will all receive notification of an event. The same or different event handlers can be used if an application loads multiple BSPs.

An application may establish as many event handlers as it wishes, for a given BSP, by calling ***BioAPI_BSPLoad*** one or more times for that BSP. An event handler is identified by a combination of address and context.

When an event occurs in a BSP, the BSP may send an event notification to the Framework by calling the Framework's event handler.

When the Framework receives an event notification from a BSP, it shall send one notification to each event handler established by each application for which that event notification is enabled for that BSP. Therefore, a single event notification callback made from a BSP to the Framework may result in zero or more callbacks made by the Framework to zero or more applications.

When the framework receives an event notification from a BSP, it shall call all the event handlers established by each application for that BSP. If an application has set up multiple event handlers, they shall be called one at a time (in any order chosen by the Framework) rather than concurrently.

Event notification may occur at any time, either during a BioAPI call (related or unrelated to the event) or while there is no BioAPI call in execution. Application writers should ensure that all callbacks are properly and safely handled by the application, no matter when the application receives them.

> NOTE: This usually requires the use of thread synchronization techniques and discipline in the actions performed by the application code placed in event handlers.

There is a "use count" on the establishment of event handlers; they have to be de-established (by ***BioAPI_BSPUnload***) as many times as they were established. When a BSP is loaded (***BioAPI_BSPLoad***), it shall raise an "insert" event immediately for each present BioAPI Unit. This will indicate to the biometric application that it can go ahead and do a ***BioAPI_BSPAttach***. If the hardware component for a specific functionality is not present, the "insert" event cannot be raised until the hardware component has been plugged in.

This function shall only be called if there is at least one call to ***BioAPI_Init*** for which a corresponding call to ***BioAPI_Terminate*** has not yet been made. The function ***BioAPI_BSPAttach*** can be invoked multiple times for each call to ***BioAPI_BSPLoad***.

The ***BioAPI_BSPLoad*** function is not to be called unless the BSP has been installed using ***BioAPI_Util_InstallBSP***. A determination of installed BSPs can be made through a call to ***BioAPI_EnumBSPs***.

8.1.5.2 Parameters

BSPUuid (input) – the UUID of the BSP selected for loading.

AppNotifyCallback (input/optional) – the event notification function provided by the caller. This defines the callback for event notifications from the loaded (and later attached) biometric service provider.

AppNotifyCallbackCtx (input) – A generic pointer to context information. When the selected biometric service provider raises an event, this value is passed as input to the event handler specified by *AppNotifyCallback*.

8.1.5.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.1.5.4 Errors

```
BioAPIERR_BSP_LOAD_FAIL
BioAPIERR_INVALID_UUID
```

See also **BioAPI Error Handling** (clause 11).

8.1.6 BioAPI_BSPUnload

```
BioAPI_RETURN BioAPI BioAPI_BSPUnload
    (const BioAPI_UUID  *BSPUuid,
    BioAPI_EventHandler  AppNotifyCallback,
    void*  AppNotifyCallbackCtx);
```

8.1.6.1 Description

The function de-registers event notification callbacks for the caller identified by *BSPUuid*. ***BioAPI_BSPUnload*** is the analogue call to ***BioAPI_BSPLoad***. If all callbacks registered with BioAPI are removed, then BioAPI unloads (for that biometric application) the BSP that was loaded by calls to ***BioAPI_BSPLoad***.

The BioAPI Framework uses the three input parameters; *BSPUuid*, *AppNotifyCallback*, and *AppNotifyCallbackCtx* to uniquely identify registered callbacks.

This function shall only be called (for a given BSP UUID) if there is at least one call to ***BioAPI_BSPLoad*** (for that BSP UUID) for which a corresponding call to this function has not yet been made.

This function should not be called by the application if there is a call to ***BioAPI_BSPAttach*** for which a corresponding call to ***BioAPI_BSPDetach*** (for a given BSP handle) has not yet been made. However, should this function be called while the BSP is still attached, then for each call to ***BioAPI_BSPAttach*** without a corresponding call to ***BioAPI_BSPDetach***, the actions relative to the missing corresponding call to ***BioAPI_BSPDetach*** shall be implicitly performed by the BioAPI Framework (as though the corresponding function had been called at that time), followed by the actions relative to ***BioAPI_BSPUnload,*** (i.e., the Framework will detach the BSP prior to unloading it).

This includes the case in which the actions relative to a missing call to ***BioAPI_BSPUnload*** are implicitly performed by the BioAPI Framework during a call to ***BioAPI_Terminate*** (see clause 8.1.2).

8.1.6.2 Parameters

BSPUuid (input) – the UUID of the BSP selected for unloading.

AppNotifyCallback (input/optional) – the event notification function to be deregistered. The function shall have been provided by the caller in ***BioAPI_BSPLoad***.

AppNotifyCallbackCtx (input/optional) – A generic pointer to context information that was provided in the corresponding call to ***BioAPI_BSPLoad***.

8.1.6.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.1.6.4 Errors

```
BioAPIERR_INVALID_UUID
BioAPIERR_BSP_NOT_LOADED
```

See also **BioAPI Error Handling** (clause 11).

8.1.7 BioAPI_BSPAttach

```
BioAPI_RETURN BioAPI BioAPI_BSPAttach
    (const BioAPI_UUID  *BSPUuid,
    BioAPI_VERSION  Version,
    const BioAPI_UNIT_LIST_ELEMENT *UnitList,
    uint32_t  NumUnits,
    BioAPI_HANDLE  *NewBSPHandle);
```

8.1.7.1 Description

This function initiates a BSP attach session and verifies that the version of the BSP expected by the application is compatible with the version on the system. The caller shall specify a list of zero or more BioAPI Units that the BSP is to use in the attach session being created.

This function shall only be called (for a given BSP UUID) if there is at least one call to ***BioAPI_BSPLoad*** (for that BSP UUID) for which a corresponding call to ***BioAPI_BSPUnload*** has not yet been made. The function ***BioAPI_BSPAttach*** can be invoked multiple times for each call to ***BioAPI_BSPLoad*** (prior to a call to ***BioAPI_BSPUnload***), for the same BSP, creating multiple invocations of that BSP.

8.1.7.2 Parameters

BSPUuid (input) – a pointer to the UUID structure containing the global unique identifier for the BSP.

Version (input) – the major and minor version number of the BioAPI specification that the application is expecting the BSP to support. The BSP shall determine whether it is compatible with the required version.

UnitList (input) – a pointer to a buffer containing a list of BioAPI_UNIT_LIST_ELEMENT structures indicating to the BSP which BioAPI Units (supported by the BSP) it is to use for this attach session. The structures contain the ID and category of each BioAPI Unit. The application may specify one of the following for each category of BioAPI Unit:

a. Select a specific BioAPI Unit: To specify a particular BioAPI Unit to use for this attach session, the ID and category of that BioAPI Unit will be provided for that element.

b. Select any BioAPI Unit: When the UnitID is set to BioAPI_DONT_CARE in a particular element, the BSP will choose which BioAPI Unit of that category to use, or will give an error return if it does not support any BioAPI Units of that category. If a particular category is not listed, the BSP will likewise choose a BioAPI Unit of that category to use if it supports a BioAPI Unit of that category (however, there is no error return if it does not).

c. Select no BioAPI Unit: When the UnitID is set to BioAPI_DONT_INCLUDE, the BSP will explicitly not attach a BioAPI Unit of the given category, even if it supports one of that category.

NOTE: Any subsequent calls requiring use of a BioAPI Unit of this category will fail with an error return.

NumUnits (input) – The number of BioAPI Unit elements in the list that the pointer *UnitList* is pointing to. If this parameter contains “0”, the BSP selects the BioAPI Unit for all categories of BioAPI Units that the BSP manages directly or indirectly.

NOTE: Only one BioAPI Unit of each category can be attached by a biometric application for each BSP attach session at any time.

NewBSPHandle (output) – a new handle that can be used to interact with the requested biometric service provider. The value will be set to BioAPIERR_FRAMEWORK_INVALID_BSP_HANDLE if the function fails.

8.1.7.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.1.7.4 Errors

```
BioAPIERR_INCOMPATIBLE_VERSION
BioAPIERR_BSP_NOT_LOADED
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_INVALID_UUID
BioAPIERR_UNIT_IN_USE
BioAPIERR_INVALID_CATEGORY
```

See also **BioAPI Error Handling** (clause 11).

8.1.8 BioAPI_BSPDetach

```
BioAPI_RETURN BioAPI BioAPI_BSPDetach
    (BioAPI_HANDLE  BSPHandle);
```

8.1.8.1 Description

This function detaches the biometric application from the BSP invocation.

At this time, all BSP allocated resources associated with the BSP attach session shall be freed or released or invalidated. This is especially important for BIR, BSP, and database handles. At this time, all set callbacks associated with the BSP attach session shall become invalid.

This function shall only be called after ***BioAPI_BSPAttach*** has been called, and shall not be called more than once for the same BSP handle created by the call to ***BioAPI_BSPAttach***.

8.1.8.2 Parameters

BSPHandle (input) – the handle that identifies the BSP attach session to be terminated.

8.1.8.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.1.8.4 Errors

`BioAPIERR_INVALID_BSP_HANDLE`

See also **BioAPI Error Handling** (clause 11).

8.1.9 BioAPI_QueryUnits

```
BioAPI_RETURN BioAPI BioAPI_QueryUnits
    (const BioAPI_UUID  *BSPUuid,
    BioAPI_UNIT_SCHEMA  **UnitSchemaArray,
    uint32_t  *NumberOfElements);
```

8.1.9.1 Description

This function provides information about all BioAPI Units that are managed directly or indirectly by the BSP identified by the given BSP UUID and are currently in the inserted state. It performs the following actions (in order):

a) determines the set of BioAPI Units that are managed directly or indirectly by the BSP and are currently in the inserted state;

b) allocates a memory block large enough to contain an array of elements of type BioAPI_UNIT_SCHEMA with as many elements as the number of BioAPI Units determined in (a);

c) fills the array with the unit schemas of all BioAPI Units determined in (a); and

d) returns the address of the array in the *UnitSchemaArray* parameter and the size of the array in the *NumberOfElements* parameter.

NOTE: When the Framework calls the function ***BioSPI_QueryUnits*** of a BSP, the BSP allocates memory for the data to be returned to the Framework. In some implementation architectures, the Framework will simply pass to the application the data and the addresses exactly as returned by the BSP because the application will interpret the addresses in the same way as the BSP and will be able to access the data that the BSP has placed at those addresses. In other implementation architectures, the framework will need to move all the data returned by the BSP to newly allocated memory blocks accessible to the application, and will call ***BioSPI_Free*** after copying each memory block but before returning from the ***BioAPI_QueryUnits*** call. In the former case, when the application calls ***BioAPI_Free***, the Framework will make a corresponding call to ***BioSPI_Free***. In the latter case, the calls to ***BioAPI_Free*** will be handled internally by the framework. However, such differences in the behavior of the Framework are not visible to the application.

The memory block containing the array shall be freed by the application via a call to ***BioAPI_Free*** (see clause 8.7.2) when it is no longer needed by the application. The memory block pointed to by the *UnitProperties* member within each element of the array shall also be freed by the application via a call to ***BioAPI_Free*** when it is no longer needed by the application.

This function shall only be called after ***BioAPI_Load*** has been called for the specified BSP, and shall not be called after ***BioAPI_Unload*** has been called for the BSP.

8.1.9.2 Parameters

BSPUuid (input) – The unique identifier for the BSP for which the unit information is to be returned.

UnitSchemaArray (output) – A pointer to the address of the array of elements of type BioAPI_UNIT_SCHEMA (allocated by the BSP - but see note above) containing the unit schema information.

NumberOfElements (output) – A pointer to the number of elements in the array.

8.1.9.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.1.9.4 Errors

```
BioAPIERR_BSP_NOT_LOADED
BioAPIERR_INVALID_UUID
```

See also **BioAPI Error Handling** (clause 11).

8.1.10 BioAPI_EnumBFPs

```
BioAPI_RETURN BioAPI BioAPI_EnumBFPs
    (BioAPI_BFP_SCHEMA  **BFPSchemaArray,
    uint32_t  *NumberOfElements);
```

8.1.10.1 Description

This function provides information about all BFPs currently installed in the component registry. It performs the following actions (in order):

a) allocates a memory block large enough to contain an array of elements of type BioAPI_BFP_SCHEMA with as many elements as the number of installed BFPs;

b) fills the array with the BFP schemas of all installed BFPs; and

c) returns the address of the array in the *BFPSchemaArray* parameter and the number of elements of the array in the *NumberOfElements* parameter.

This function shall only be called if there is at least one call to ***BioAPI_Init*** for which a corresponding call to ***BioAPI_Terminate*** has not yet been made.

This function is handled internally within the BioAPI Framework and is not passed through to any BSP.

The memory block containing the array shall be freed by the application via a call to ***BioAPI_Free*** (see clause 8.7.2) when it is no longer needed by the application.

The memory blocks pointed to by the *Path* and *BFPProperty* members within each element of the array shall also be freed by the application via a call to ***BioAPI_Free*** when they are no longer needed by the application.

8.1.10.2 Parameters

BFPSchemaArray (output) – A pointer to the address of the array of elements of type BioAPI_BFP_SCHEMA (allocated by the framework) containing the BFP schema information.

NumberOfElements (output) – A pointer to the number of elements of the array (which is also the number of BFP schemas in the component registry).

8.1.10.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.1.10.4 Errors

See **BioAPI Error Handling** (clause 11).

8.1.11 BioAPI_QueryBFPs

```
BioAPI_RETURN BioAPI BioAPI_QueryBFPs
    (const BioAPI_UUID  *BSPUuid,
    BioAPI_BFP_LIST_ELEMENT  **BFPList,
    uint32_t  *NumberOfElements);
```

8.1.11.1 Description

This function returns a list of BFPs which are currently installed in the component registry and supported by the BSP identified by the given BSP UUID. It performs the following actions (in order):

a) determines which among all currently installed BFPs are supported by the BSP;

b) allocates a memory block large enough to contain an array of elements of type BioAPI_BFP_LIST_ELEMENT with as many elements as the number of BFPs determined in (a);

c) fills the array with identification information (category and UUID) about the BFPs determined in (a); and

d) returns the address of the array in the *BFPList* parameter and the number of elements of the array in the *NumberOfElements* parameter.

NOTE: When the Framework calls the function ***BioSPI_QueryBFPs*** of a BSP, the BSP allocates memory for the data to be returned to the Framework. In some implementation architectures, the Framework will simply pass to the application the data and the addresses exactly as returned by the BSP because the application will interpret the addresses in the same way as the BSP and will be able to access the data that the BSP has placed at those addresses. In other implementation architectures, the framework will need to move all the data returned by the BSP to newly allocated memory blocks accessible to the application, and call ***BioSPI_Free*** after copying each memory block but before returning from the ***BioAPI_QueryBFPs*** call. In the former case, when the application calls ***BioAPI_Free***, the Framework will make a corresponding call to ***BioSPI_Free***. In the latter case, the calls to ***BioAPI_Free*** will be handled internally by the framework. However, such differences in the behavior of the Framework are not visible to the application.

Additional information about the supported BFPs can be retrieved by calling ***BioAPI_EnumBFPs*** and analyzing the *BFPSchemaArray* at the matching *BFPUuid*s.

This function shall only be called after ***BioAPI_Load*** has been called for the specified BSP, and shall not be called after ***BioAPI_Unload*** has been called for the BSP.

The memory block containing the array shall be freed by the application via a call to ***BioAPI_Free*** (see clause 8.7.2) when it is no longer needed by the application.

8.1.11.2 Parameters

BSPUuid (input) – The unique identifier for the BSP for which the BFP information is to be returned.

BFPList (output) – A pointer to he address of the array of elements of type BioAPI_BFP_LIST_ELEMENT (allocated by the BSP - but see note above), a buffer containing the BFP identification information.

NumberOfElements (output) – A pointer to the number of elements in the array.

8.1.11.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.1.11.4 Errors

`BioAPIERR_INVALID_BSP_HANDLE`

See also **BioAPI Error Handling** (clause 11).

8.1.12 BioAPI_ControlUnit

```
BioAPI_RETURN BioAPI BioAPI_ControlUnit
    (BioAPI_HANDLE  BSPHandle,
     BioAPI_UNIT_ID  UnitID,
     uint32_t  ControlCode,
     const BioAPI_DATA  *InputData,
     BioAPI_DATA  *OutputData);
```

8.1.12.1 Description

This function sends control data from the application to the BioAPI Unit and receives status or operation data from that unit. The content of the *ControlCode*, the send (input) data, and the receive (output) data will be specified in the related interface specificationfor this BioAPI Unit (or associated FPI, if present).

The function allocates a memory block large enough to contain the output data that are to be returned to the application, fills the memory block with the data, and sets the fields *Length* and *Data* of the *OutputData* structure to the size and address of the memory block (respectively).

The memory block returned by the BioAPI function call shall be freed by the application using ***BioAPI_Free*** (see clause 8.7.2).

8.1.12.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

UnitID (input) – ID of the BioAPI Unit.

ControlCode (input) – The function code in the BioAPI Unit to be called.

InputData (input) – A pointer to a BioAPI-DATA structure containing an address and length of a buffer containing the data to be sent to the BioAPI Unit related to the given *ControlCode*.

OutputData (output) – Pointer to a BioAPI_DATA structure. On output, this shall contain the address and length of a data buffer containing the data received from the BioAPI Unit after processing the function indicated by the *ControlCode*. If no memory block has been allocated by the function, the address shall be set to NULL and the length shall be set to zero.

8.1.12.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.1.12.4 Errors

```
BioAPIERR_BIOAPI_UNIT_NOT_INSERTED
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_UNIT_IN_USE
BioAPIERR_INVALID_BSP_HANDLE
```

See also **BioAPI Error Handling** (clause 11).

8.2 Data Handle Operations

8.2.1 BioAPI_FreeBIRHandle

```
BioAPI_RETURN BioAPI BioAPI_FreeBIRHandle
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_BIR_HANDLE  Handle);
```

8.2.1.1 Description

This function frees the memory and resources associated with the specified BIR Handle. The associated BIR is no longer referenceable through that handle. If necessary, the biometric application can store the BIR into a BSP-controlled BIR database (using ***BioAPI_DbStoreBIR***) before calling ***BioAPI_FreeBIRHandle***. Alternatively, the application can call ***BioAPI_GetBIRFromHandle*** (which will retrieve the BIR and free the handle) instead of calling ***BioAPI_FreeBIRHandle***.

This function shall only be called after ***BioAPI_BSPAttach*** has been called, and shall not be called after ***BioAPI_BSPDetach*** has been called for the BSP handle created by the call to ***BioAPI_BSPAttach***.

8.2.1.2 Parameters

BSPHandle (input) – The handle of an attached BSP.

Handle (input) – The BIR handle to be freed.

8.2.1.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.2.1.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BSP_HANDLE
```

See also **BioAPI Error Handling** (clause 11).

8.2.2 BioAPI_GetBIRFromHandle

```
BioAPI_RETURN BioAPI BioAPI_GetBIRFromHandle
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_BIR_HANDLE  Handle,
    BioAPI_BIR  *BIR);
```

8.2.2.1 Description

This function returns the BIR associated with a BIR handle returned by a BSP. The BIR handle is freed by the BSP before the function returns.

8.2.2.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

Handle (input) – The handle of the BIR to be retrieved.

BIR (output) – Pointer to the retrieved BIR.

8.2.2.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.2.2.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BSP_HANDLE
```

See also **BioAPI Error Handling** (clause 11).

8.2.3 BioAPI_GetHeaderFromHandle

```
BioAPI_RETURN BioAPI BioAPI_GetHeaderFromHandle
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_BIR_HANDLE  Handle,
    BioAPI_BIR_HEADER  *Header);
```

8.2.3.1 Description

This function retrieves the BIR header of the BIR identified by *Handle*. The BIR handle is not freed by the BSP.

8.2.3.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

Handle (input) – The handle of the BIR whose header is to be retrieved.

Header (output) – The header of the specified BIR.

8.2.3.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.2.3.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BSP_HANDLE
```

See also **BioAPI Error Handling** (clause 11).

8.3 Callback and Event Operations

8.3.1 BioAPI_EnableEvents

```
BioAPI_RETURN BioAPI BioAPI_EnableEvents
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_EVENT_MASK  Events);
```

8.3.1.1 Description

This function enables the events specified by the Event Mask coming from all the BioAPI Units selected in the BSP attach session identified by the BSP Handle, and disables all other events from those BioAPI Units. Events from other BioAPI Units directly or indirectly managed by the same BSP (possibly selected in other attach sessions but not selected in the specified attach session) are not affected.

In some cases, INSERT events cannot be suppressed (see clause 7.27).

8.3.1.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

Events (input) – A mask indicating which events to enable.

8.3.1.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.3.1.4 Errors

See **BioAPI Error Handling** (clause 11).

8.3.2 BioAPI_SetGUICallbacks

```
BioAPI_RETURN BioAPI BioAPI_SetGUICallbacks
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_GUI_STREAMING_CALLBACK  GuiStreamingCallback,
    void  *GuiStreamingCallbackCtx,
    BioAPI_GUI_STATE_CALLBACK  GuiStateCallback,
    void  *GuiStateCallbackCtx);
```

8.3.2.1 Description

This function allows the application to establish callbacks so that the application may control the "look-and-feel" of the biometric user interface by receiving from the BSP a sequence of bit-map images, called streaming data, for display by the biometric application as well as state information.

NOTE: Not all BSPs support the provision of streaming data.

8.3.2.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

GuiStreamingCallback (input) – A pointer to an application callback to deal with the presentation of biometric streaming data.

GuiStreamingCallbackCtx (input) – A generic pointer to context information that was provided by the application and will be presented on the callback.

GuiStateCallback (input) – A pointer to an application callback to deal with GUI state changes.

GuiStateCallbackCtx (input) – A generic pointer to context information that was provided by the application and will be presented on the callback.

NOTE 1: Function sub-types for BioAPI_GUI_STATE_CALLBACK and BioAPI_GUI_STREAMING_CALLBACK are defined in clauses 7.36 and 7.37 respectively.

NOTE 2: See also clause C.7 on User Interface Considerations.

8.3.2.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.3.2.4 Errors

```
BioAPIERR_INVALID_POINTER
BioAPIERR_INVALID_BSP_HANDLE
```

See also **BioAPI Error Handling** (clause 11).

8.4 Biometric Operations

8.4.1 BioAPI_Capture

```
BioAPI_RETURN BioAPI BioAPI_Capture
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_BIR_PURPOSE  Purpose,
    BioAPI_BIR_SUBTYPE  Subtype,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
    BioAPI_BIR_HANDLE  *CapturedBIR,
    int32_t  Timeout,
    BioAPI_BIR_HANDLE  *AuditData);
```

8.4.1.1 Description

This function captures samples for the purpose specified, and the BSP returns either an "intermediate" type BIR (if the ***BioAPI_Process*** function needs to be called), or a "processed" BIR (if not). The *Purpose* is recorded in the header of the *CapturedBIR*. If *AuditData* is non-NULL, a BIR of type "raw" may be returned. The function returns handles to whatever data is collected, and all local operations can be completed through use of the handles. If the application needs to acquire the data either to store it in a BIR database or to send it to a server, the application can retrieve the data with the ***BioAPI_GetBIRFromHandle*** function.

By default, the BSP is responsible for providing the user interface associated with the capture operation. The application may, however, request control of the GUI "look-and-feel" by providing a GUI callback pointer in ***BioAPI_SetGUICallbacks***. See clause C.7 for additional explanation of user interface features.

Capture serializes use of the sensor device. If two or more biometric applications are racing for the sensor, the losers will wait until the operation completes or the timeout expires. This serialization takes place in all functions that capture data. The BSP is responsible for serializing. It may do this by either returning 'busy' (BioAPI_UNIT_IN_USE) or by queuing requests.

The BIR Handle returned by the function shall be released by the application (via ***BioAPI_FreeBIRHandle***) when it is no longer needed.

8.4.1.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

Purpose (input) – A value indicating the purpose of the biometric data capture.

Subtype (input/optional) – Specifies which subtype (e.g., left/right eye) to capture. A value of BioAPI_NO_SUBTYPE_AVAILABLE (0x00) indicates that the BSP is to select the subtype(s).

NOTE: Not all BSPs support the capture of specific Subtypes. Actual Subtypes captured will be reflected in the header of the returned *CapturedBIR*.

OutputFormat (input/optional) – Specifies which BDB format to use for the returned *CapturedBIR*, if the BSP supports more than one format. A NULL pointer value indicates that the BSP is to select the format.

CapturedBIR (output) – A handle to a BIR containing captured data. This data is either an "intermediate" type BIR (which can only be used by either the ***BioAPI_Process,*** ***BioAPI_CreateTemplate***, or ***BioAPI_ProcessWithAuxData*** functions, depending on the purpose), or a "processed" BIR (which can be used directly by ***BioAPI_VerifyMatch*** or ***BioAPI_IdentifyMatch***, depending on the purpose).

Timeout (input) – An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive number. A '-1' value means the BSP's default timeout value will be used.

AuditData (output/optional) – A handle to a BIR containing raw biometric data. This data may be used to provide human-identifiable data of the person at the sensor unit. If the pointer is NULL on input, no audit data is collected. Not all BSPs support the collection of audit data. A BSP may return a BIR handle value of BioAPI_UNSUPPORTED_BIR_HANDLE to indicate *AuditData* is not supported, or a value of BioAPI_INVALID_BIR_HANDLE to indicate that no audit data is available.

8.4.1.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.4.1.4 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_UNABLE_TO_CAPTURE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_TIMEOUT_EXPIRED
BioAPIERR_PURPOSE_NOT_SUPPORTED
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_UNIT_IN_USE
```

See also **BioAPI Error Handling** (clause 11).

8.4.2 BioAPI_CreateTemplate

```
BioAPI_RETURN BioAPI BioAPI_CreateTemplate
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_INPUT_BIR  *CapturedBIR,
    const BioAPI_INPUT_BIR  *ReferenceTemplate,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *OutputFormat,
    BioAPI_BIR_HANDLE  *NewTemplate,
    const BioAPI_DATA  *Payload,
    BioAPI_UUID  *TemplateUUID);
```

8.4.2.1 Description

This function takes a BIR containing biometric data in intermediate form for the purpose of creating a new enrollment template. A new BIR is constructed from the *CapturedBIR*, and (optionally) it may perform an update based on an existing *ReferenceTemplate*. The old *ReferenceTemplate* remains unchanged.

The optional input *ReferenceTemplate* is provided for use in creating the *NewTemplate*, if the BSP supports this capability. When present, use of the input *ReferenceTemplate* by the BSP to create the output *NewTemplate* is optional.

If the BSP supports an internal or BSP-controlled BIR database (e.g., smartcard or identification engine), it may optionally return the *UUID* assigned to the newly created *NewTemplate* as stored within that BSP-controlled BIR database. The UUID value shall be the same as that included in the BIR header, if present.

The BIR handle returned by the function shall be released by the application (via ***BioAPI_FreeBIRHandle***) when it is no longer needed. The BIR may be retrieved by calling ***BioAPI_GetBIRFromHandle***, which also releases the handle.

8.4.2.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

CapturedBIR (input) – The captured BIR or its handle.

ReferenceTemplate (input/optional) – Optionally, the existing template to be updated, or its key in a BIR database, or its handle or NULL if a reference template does not exist yet or is not intended to be updated.

OutputFormat (input/optional) – Specifies which BDB format to use for the returned *NewTemplate*, if the BSP supports more than one format. A NULL pointer value indicates that the BSP is to select the format.

NewTemplate (output) – A handle to a newly created template that is derived from the *CapturedBIR* and (optionally) the *ReferenceTemplate*.

Payload (input/optional) – A pointer to data that will be stored by the BSP. This parameter is ignored if NULL.

NOTE 1: Not all BSPs support storage of payloads.

NOTE 2: See clauses A.4.6.2.6 and C.5 for additional information regarding payloads.

TemplateUUID (output/optional) – A pointer to a 16-byte memory block where the BSP-assigned UUID associated with the *ReferenceTemplate* (as stored internally in a BSP-controlled BIR database) will optionally be returned. The pointer shall be set to NULL to indicate that no UUID is being returned.

8.4.2.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.4.2.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BIR
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_UNABLE_TO_STORE_PAYLOAD
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_PURPOSE_NOT_SUPPORTED
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
```

See also **BioAPI Error Handling** (clause 11).

8.4.3 BioAPI_Process

```
BioAPI_RETURN BioAPI BioAPI_Process
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_INPUT_BIR  *CapturedBIR,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *OutputFormat,
    BioAPI_BIR_HANDLE  *ProcessedBIR);
```

8.4.3.1 Description

This function processes the intermediate data captured via a call to ***BioAPI_Capture*** for the purpose of either verification or identification. If the processing capability is supported by the attached BSP invocation, the BSP builds a "processed biometric sample" BIR; otherwise, *ProcessedBIR* is set to NULL, and this function returns BioAPIERR_BSP_FUNCTION_NOT_SUPPORTED.

This function results in the creation of a BIR by the BSP. The application can retrieve the BIR using the BIR handle through a call to ***BioAPI_GetBIRFromHandle***, which also frees the handle, or can release the memory associated with the BIR handle only through a call to ***BioAPI_FreeBIRHandle***.

8.4.3.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

CapturedBIR (input) – The captured BIR or its handle.

OutputFormat (input/optional) – Specifies which BDB format to use for the returned *ProcessedBIR*, if the BSP supports more than one format. A NULL pointer value indicates that the BSP is to select the format.

ProcessedBIR (output) – A handle for the newly constructed "processed" BIR.

8.4.3.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.4.3.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BIR
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_PURPOSE_NOT_SUPPORTED
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_QUALITY_ERROR
```

See also **BioAPI Error Handling** (clause 11).

8.4.4 BioAPI_ProcessWithAuxBIR

```
BioAPI_RETURN BioAPI BioAPI_ProcessWithAuxBIR
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_INPUT_BIR  *CapturedBIR,
    const BioAPI_INPUT_BIR  *AuxiliaryData,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *OutputFormat,
    BioAPI_BIR_HANDLE  *ProcessedBIR);
```

8.4.4.1 Description

This function processes the intermediate data previously captured via a call to ***BioAPI_Capture*** in conjunction with auxiliary data, creating processed biometric samples for the purpose of subsequent verification or identification. It enables implementations that require the input of auxiliary data to the process operation.

NOTE: This capability may be used to support biometric match-on-card (MOC). See clause C.8 for a description of BioAPI use within the overall MOC process.

If the processing with auxiliary data capability is supported by the attached BSP invocation, the BSP builds a "processed biometric sample" BIR, otherwise, *ProcessedBIR* is set to NULL, and this function returns BioAPIERR_BSP_FUNCTION_NOT_SUPPORTED.

This function results in the creation of a BIR by the BSP. The application can retrieve the BIR using the BIR handle through a call to ***BioAPI_GetBIRFromHandle***, which also frees the handle, or can release the memory associated with the BIR handle only through a call to ***BioAPI_FreeBIRHandle***.

8.4.4.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

CapturedBIR (input) – A BIR obtained from the ***BioAPI_Capture*** function previously called.

AuxiliaryData (input) – A BIR structure containing auxiliary data used in the processing operation.

NOTE 1: An example of auxiliary data is information related to the enrollment template which allows the processing operation to properly crop an input image to maximize the possibility of a subsequent match (e.g., to ensure that the processed BIR for verification and the enrollment template are derived from the same portion of a finger). Another would be passing of biometric algorithm parameters.

NOTE 2: The content and format of the auxiliary data are specified by the BIR Biometric Data Format field in the auxiliary BIR header and may be specific to a BSP.

OutputFormat (input/optional) – Specifies which BDB format to use for the returned *ProcessedBIR*, if the BSP supports more than one format. A NULL pointer value indicates that the BSP is to select the format.

ProcessedBIR (output) – A handle for the newly constructed "processed" BIR.

8.4.4.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.4.4.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BIR
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TOO_MANY_HANDLES
```

```
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_PURPOSE_NOT_SUPPORTED
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_FUNCTION_NOT_SUPPORTED
```

See also **BioAPI Error Handling** (clause 11).

8.4.5 BioAPI_VerifyMatch

```
BioAPI_RETURN BioAPI BioAPI_VerifyMatch
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_FMR  MaxFMRRequested,
    const BioAPI_INPUT_BIR  *ProcessedBIR,
    const BioAPI_INPUT_BIR  *ReferenceTemplate,
    BioAPI_BIR_HANDLE  *AdaptedBIR,
    BioAPI_BOOL  *Result,
    BioAPI_FMR  *FMRAchieved,
    BioAPI_DATA  *Payload);
```

8.4.5.1 Description

This function performs a verification (1-to-1) match between two BIRs: the *ProcessedBIR* and the *ReferenceTemplate*. The *ProcessedBIR* is the "processed" BIR constructed specifically for this verification. The *ReferenceTemplate* was created at enrollment.

The application shall request a maximum FMR value criterion (threshold) for a successful match. The Boolean *Result* indicates whether verification was successful or not, and the *FMRAchieved* is a FMR value (score) indicating how closely the BIRs actually matched.

NOTE: See clause C.4 for information on the use of the FMR concept for normalized scoring and thresholding.

By setting the *AdaptedBIR* pointer to an address other than NULL, the application can request that a BIR be constructed by adapting the *ReferenceTemplate* using the *ProcessedBIR*. A new handle is returned to the *AdaptedBIR*. If the match is successful, an attempt may be made to adapt the *ReferenceTemplate* with information taken from the *ProcessedBIR*. (Not all BSPs perform adaptation). The resulting *AdaptedBIR* should now be considered an optimal enrollment template, and be saved in the BIR database. (It is up to the application whether it uses or discards this data.) It is important to note that adaptation may not occur in all cases. In the event of an adaptation, this function stores the handle to the new BIR in the memory pointed to by the *AdaptedBIR* parameter.

If a *Payload* is associated with the *ReferenceTemplate*, the *Payload* may be returned upon successful verification if the *FMRAchieved* is sufficiently stringent; this is controlled by the policy of the BSP and specified in its schema.

NOTE 1: Not all BSPs support return of payloads.

NOTE 2: See clauses A.4.6.2.6 and C.5 for additional information regarding use of payloads.

The memory block returned by the BioAPI function call shall be freed by the application as soon as it is no longer needed using ***BioAPI_Free*** (see clause 8.7.2). If an adapted BIR is returned, its handle can be released through a call to ***BioAPI_FreeBIRHandle***.

8.4.5.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

MaxFMRRequested (input) – The requested FMR criterion for successful verification (i.e., the matching threshold).

ProcessedBIR (input) – The BIR to be verified, or its handle.

ReferenceTemplate (input) – The BIR to be verified against, or its key in a BIR database, or its handle.

AdaptedBIR (output/optional) – A pointer to the handle of the adapted BIR. This parameter can be NULL if an adapted BIR is not desired. Not all BSPs support the adaptation of BIRs. The function may

return a handle value of BioAPI_UNSUPPORTED_BIR_HANDLE to indicate that adaptation is not supported or a value of BioAPI_INVALID_BIR_HANDLE to indicate that adaptation was not possible.

Result (output) – A pointer to a Boolean value indicating (BioAPI_TRUE/BioAPI_FALSE) whether the BIRs matched or not according to the specified criteria.

FMRAchieved (output) – A pointer to an FMR value indicating the closeness of the match (i.e., the match score).

Payload (output/optional) – If a payload is associated with the *ReferenceTemplate*, it is returned in an allocated BioAPI_DATA structure if the *FMRAchieved* satisfies the payload policy of the BSP.

8.4.5.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.4.5.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BIR
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_BIR_NOT_FULLY_PROCESSED
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
```

See also **BioAPI Error Handling** (clause 11).

8.4.6 BioAPI_IdentifyMatch

```
BioAPI_RETURN BioAPI BioAPI_IdentifyMatch
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_FMR  MaxFMRRequested,
    const BioAPI_INPUT_BIR  *ProcessedBIR,
    const BioAPI_IDENTIFY_POPULATION  *Population,
    uint32_t  TotalNumberOfTemplates,
    BioAPI_BOOL  Binning,
    uint32_t  MaxNumberOfResults,
    uint32_t  *NumberOfResults,
    BioAPI_CANDIDATE  **Candidates,
    int32_t Timeout);
```

8.4.6.1 Description

This function performs an identification (1-to-many) match between a *ProcessedBIR* and a set of reference BIRs. The *ProcessedBIR* is the "processed" BIR captured specifically for this identification. The population that the match takes place against can be presented in one of two ways:

a) In a BIR database identified by an open database handle;

b) Input in an array of BIRs;

NOTE: When using a BSP-controlled BIR database, this database must previously have been opened using ***BioAPI_DbOpen***.

There is an option to use an array of BIRs, which can be specified in BioAPI_IDENTIFY_POPULATION_TYPE in the BioAPI_IDENTIFY_POPULATION structure. If it is specified as BioAPI_PRESET_ARRAY_TYPE (3), the array of BIRs which had been previously set in the ***BioAPI_PresetIdentifyPopulation*** call will be used. The preset array of BIRs will be freed internally by the BSP when ***BioAPI_BSPDetach*** is called.

The function performs the following actions (in order):

a) determines the set of candidates from the population that match according to the specified criteria;

b) allocates a memory block large enough to contain an array of elements of type BioAPI_CANDIDATE with as many elements as the number of candidates determined in (a);

c) fills the array with the candidate information for all candidates determined in (a), including the *FMRAchieved* of each candidate; and

d) returns the address of the array in the *Candidates* parameter and the size of the array in the *NumberOfResults* parameter.

NOTE: See clause C.4 for information on the use of the FMR concept for normalized scoring and thresholding.

The memory block returned by the BioAPI function call shall be freed by the application using ***BioAPI_Free*** (see clause 8.7.2).

8.4.6.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

MaxFMRRequested (input) – The requested FMR criterion for successful identification (i.e., the matching threshold).

ProcessedBIR (input) – The BIR to be identified.

Population (input) – The population of reference BIRs (templates) against which the identification is to be performed (by this BSP).

TotalNumberOfTemplates (input) – Specifies the total number of templates stored by the application in the population. A value of zero indicates that the application is not providing a number.

NOTE: If the total population is distributed over several databases/partitions, then the total size of the population will be greater than the population seen by the BSP. The BSP may map the *FARRequested* to its internal matching threshold based on this total population size.

Binning (input) – A Boolean indicating whether *Binning* is on or off.

NOTE 1: Binning is a search optimization technique that the BSP may employ. It is based on searching a subset of the population according to the intrinsic characteristics of the biometric data. While it may improve the speed of the Match operation, it may also increase the probability of missing a candidate (due to the possibility of mis-binning and as a result, searching a bin which should, but does not, contain the matching BIR).

NOTE 2: Additional information regarding binning is located in clause A.4.6.2.10.

MaxNumberOfResults (input) – Specifies the maximum number of match candidates to be returned as a result of the 1:N match. A value of zero is a request for all candidates.

NumberOfResults (output) – A pointer to the number of candidates returned in the *Candidates* array as a result of the 1:N match.

Candidates (output) – A pointer to the address of an array of elements of type BioAPI_CANDIDATE containing information about the BIRs identified as a result of the match process (i.e., indices associated with BIRs found to meet the match threshold). This list is in rank order, with the best scoring (closest matching) record being first. If no matches are found, no array is allocated and a NULL address is returned. If the *Population* was presented in a BIR database (i.e., the value of BioAPI_IDENTIFY_POPULATION_TYPE is BioAPI_DB_TYPE), the array contains pointers to UUID's corresponding to BIR's stored in the BSP-controlled BIR database. If the *Population* was presented as a passed-in array of BIRs, then BioAPI_IDENTIFY_POPULATION_TYPE has the value BioAPI_ARRAY_TYPE and the array contains pointers to relative indices into the passed-in array.

Timeout (input) – An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, no array is allocated, and a NULL address is returned. This value can be any positive number. A '-1' value means the BSP's default timeout value will be used.

8.4.6.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.4.6.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TIMEOUT_EXPIRED
BioAPIERR_NO_INPUT_BIRS
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_BIR_NOT_FULLY_PROCESSED
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_FUNCTION_FAILED
BioAPIERR_PRESET_BIR_DOES_NOT_EXIST
BioAPIERR_INVALID_DB_HANDLE
```

See also **BioAPI Error Handling** (clause 11).

NOTE 1: Not all BSPs support 1:N identification. See your BSP programmer's manual to determine if the BSP(s) you are using supports this capability.

NOTE 2: Depending on the BSP and the location and size of the database to be searched, this operation could take a significant amount of time to perform. Check your BSP manual for recommended *Timeout* values.

NOTE 3: The number of match candidates found by the BSP is dependent on the actual FMR of the matching algorithm at the *MaxFMRRequested* threshold setting.

8.4.7 BioAPI_Enroll

```
BioAPI_RETURN BioAPI BioAPI_Enroll
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_BIR_PURPOSE  Purpose,
    BioAPI_BIR_SUBTYPE  Subtype,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
    const BioAPI_INPUT_BIR  *ReferenceTemplate,
    BioAPI_BIR_HANDLE  *NewTemplate,
    const BioAPI_DATA  *Payload,
    int32_t  Timeout,
    BioAPI_BIR_HANDLE  *AuditData,
    BioAPI_UUID  *TemplateUUID);
```

8.4.7.1 Description

This function captures biometric data from the attached device (sensor unit) for the purpose of creating a *ProcessedBIR* for the purpose of BioAPI_PURPOSE_ENROLL, BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY, or BioAPI_PURPOSE_ENROLL_FOR_IDENTIFICATION ONLY (i.e., a reference template).

The optional input *ReferenceTemplate* is provided for use in creating the *NewTemplate*, if the BSP supports the template update capability. When present, use of the input *ReferenceTemplate* by the BSP to create the output *NewTemplate* is optional.

If the BSP supports an internal (or BSP-controlled) BIR database (e.g., smartcard or identification engine), it may optionally return the *UUID* assigned to the newly created *ReferenceTemplate* as stored within that BSP-controlled BIR database. The UUID value shall be the same as that included in the BIR header, if present.

The BSP is responsible for providing the user interface associated with the enrollment operation as a default. The application may request control of the GUI "look-and-feel" by providing a GUI callback pointer in ***BioAPI_SetGUICallbacks***. See clause C.7 for additional explanation of user interface features.

Since the ***BioAPI_Enroll*** operation includes a capture, it serializes use of the sensor device. If two or more applications are racing for the device, the losers will wait until the operation completes or the timeout expires. This serialization takes place in all functions that capture data. The BSP is responsible for serializing. It may do this by either returning 'busy' (BioAPI_UNIT_IN_USE) or by queuing requests.

The memory block returned by the BioAPI function call shall be freed by the application as soon as it is no longer needed using ***BioAPI_Free*** (see clause 8.7.2). Output BIRs can be retrieved by a call to ***BioAPI_GetBIRFromHandle***, which releases the handle, or the handle can be released without retrieving the BIR through ***BioAPI_FreeBIRHandle***.

8.4.7.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

Purpose (input) – A value indicating the desired purpose (BioAPI_PURPOSE_ENROLL, BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY, or BioAPI_PURPOSE_ENROLL_FOR_IDENTIFICATION ONLY).

Subtype (input/optional) – Specifies which subtype (e.g., left/right eye) to enroll. A value of BioAPI_NO_SUBTYPE_AVAILABLE (0x00) indicates that the BSP is to select the subtype(s).

NOTE: Not all BSPs support the capture of specific Subtypes. Actual Subtypes captured will be reflected in the header of the returned *NewTemplate*.

OutputFormat (input/optional) – Specifies which BDB format to use for the returned *NewTemplate*, if the BSP supports more than one format. A NULL pointer value indicates that the BSP is to select the format.

ReferenceTemplate (input/optional) – Optionally, the BIR to be adapted (updated), or it's key in a BIR database, or its handle.

NewTemplate (output) – A handle to a newly created template that is derived from the new raw samples and (optionally) the *ReferenceTemplate*.

Payload (input/optional) – A pointer to data that will be stored by the BSP. This parameter is ignored if NULL.

NOTE 1: Not all BSPs support storage of payloads.

NOTE 2: See clauses A.4.6.2.6 and C.5 for additional information regarding payloads.

Timeout (input) – An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive number. A '-1' value means the BSP's default timeout value will be used.

AuditData (output/optional) – A handle to a BIR containing raw biometric data. This data may be used to provide a human-identifiable data of the person at the sensor unit. If the pointer is NULL on input, no audit data is collected. Not all BSPs support the collection of audit data. A BSP may return a BIR handle value of BioAPI_UNSUPPORTED_BIR_HANDLE to indicate *AuditData* is not supported, or a value of BioAPI_INVALID_BIR_HANDLE to indicate that no audit data is available.

TemplateUUID (output/optional) – A pointer to a 16-byte memory block where the BSP-assigned UUID associated with the *ReferenceTemplate* (as stored internally in a BSP-controlled BIR database) will optionally be returned. The pointer shall be set to NULL to indicate that no UUID is being returned.

8.4.7.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.4.7.4 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_UNABLE_TO_CAPTURE
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_UNABLE_TO_STORE_PAYLOAD
BioAPIERR_TIMEOUT_EXPIRED
BioAPIERR_PURPOSE_NOT_SUPPORTED
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_UNIT_IN_USE
```

See also **BioAPI Error Handling** (clause 11).

8.4.8 BioAPI_Verify

```
BioAPI_RETURN BioAPI BioAPI_Verify
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_FMR  MaxFMRRequested,
    const BioAPI_INPUT_BIR  *ReferenceTemplate,
    BioAPI_BIR_SUBTYPE  Subtype,
    BioAPI_BIR_HANDLE  *AdaptedBIR,
    BioAPI_BOOL  *Result,
    BioAPI_FMR  *FMRAchieved,
    BioAPI_DATA  *Payload,
    int32_t  Timeout,
    BioAPI_BIR_HANDLE  *AuditData);
```

8.4.8.1 Description

This function captures biometric data from the attached device (sensor unit), and compares it against the *ReferenceTemplate*.

The application shall request a maximum FMR value criterion (threshold) for a successful match. The Boolean *Result* indicates whether verification was successful or not, and the *FMRAchieved* is a FMR value (score) indicating how closely the BIRs actually matched.

NOTE: See clause C.4 for information on the use of the FMR concept for normalized scoring and thresholding.

By setting the *AdaptedBIR* pointer to non-NULL, the application can request that a BIR be constructed by adapting the *ReferenceTemplate* using the *ProcessedBIR*. A new handle is returned to the *AdaptedBIR.* If the match is successful, an attempt may be made to adapt the *ReferenceTemplate* with information taken from the *ProcessedBIR*. (Not all BSPs perform adaptation). The resulting *AdaptedBIR* should now be considered an optimal enrollment template, and be saved in the BIR database. (It is up to the application whether it uses or discards this data). It is important to note that adaptation may not occur in all cases. In the event of an adaptation, this function stores the handle to the new BIR in the memory pointed to by the *AdaptedBIR* parameter.

If a *Payload* is associated with the *ReferenceTemplate*, the *Payload* may be returned upon successful verification if the *FMRAchieved* is sufficiently stringent; this is controlled by the policy of the BSP and specified in its schema.

NOTE 1: Not all BSPs support return of payloads.

NOTE 2: See clauses A.4.6.2.6 and C.5 for additional information regarding use of payloads.

The BSP is responsible for providing the user interface associated with the verify operation as a default. The application may request control of the GUI “look-and-feel” by providing a GUI callback pointer in ***BioAPI_SetGUICallbacks***. See clause C.7 for additional explanation of user interface features.

Since the ***BioAPI_Verify*** operation includes a capture, it serializes use of the sensor device. If two or more applications are racing for the device, the losers will wait until the operation completes or the timeout expires. This serialization takes place in all functions that capture data. The BSP is responsible for serializing. It may do this by either returning ‘busy’ (BioAPI_UNIT_IN_USE) or by queuing requests.

The memory block returned by the BioAPI function call shall be freed by the application as soon as it is no longer needed using ***BioAPI_Free*** (see clause 8.7.2). Output BIRs can be retrieved by a call to ***BioAPI_GetBIRFromHandle***, which releases the handle, or the handle can be released without retrieving the BIR through ***BioAPI_FreeBIRHandle***.

8.4.8.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

MaxFMRRequested (input) – The requested FMR criterion for successful verification (i.e., the matching threshold).

ReferenceTemplate (input) – The BIR to be verified against, or its key in a BIR database, or its handle.

Subtype (input/optional) – Specifies which subtype (e.g., left/right eye) to capture for verification. A value of BioAPI_NO_SUBTYPE_AVAILABLE (0x00) indicates that the BSP is to select the subtype(s).

NOTE: Not all BSPs support the capture of specific Subtypes.

AdaptedBIR (output/optional) – A pointer to the handle of the adapted BIR. This parameter can be NULL if an adapted BIR is not desired. Not all BSPs support the adaptation of BIRs. The function may return a handle value of BioAPI_UNSUPPORTED_BIR_HANDLE to indicate that adaptation is not supported or a value of BioAPI_INVALID_BIR_HANDLE to indicate that adaptation was not possible.

Result (output) – A pointer to a Boolean value indicating (BioAPI_TRUE/BioAPI_FALSE) whether the BIRs matched or not according to the specified criteria.

FMRAchieved (output) – A pointer to an FMR value indicating the closeness of the match (i.e., the match score).

Payload (output/optional) – If a payload is associated with the *ReferenceTemplate*, it is returned in an allocated BioAPI_DATA structure if the *FMRAchieved* satisfies the policy of the BSP.

Timeout (input) – An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive number. A '-1' value means the BSP's default timeout value will be used.

AuditData (output/optional) – A handle to a BIR containing raw biometric data. This data may be used to provide human-identifiable data of the person at the sensor unit. If the pointer is NULL on input, no audit data is collected. Not all BSPs support the collection of audit data. A BSP may return a BIR handle value of BioAPI_UNSUPPORTED_BIR_HANDLE to indicate *AuditData* is not supported, or a value of BioAPI_INVALID_BIR_HANDLE to indicate that no audit data is available.

8.4.8.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.4.8.4 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_UNABLE_TO_CAPTURE
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_TIMEOUT_EXPIRED
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_UNIT_IN_USE
```

See also **BioAPI Error Handling** (clause 11).

8.4.9 BioAPI_Identify

```
BioAPI_RETURN BioAPI BioAPI_Identify
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_FMR  MaxFMRRequested,
    BioAPI_BIR_SUBTYPE  Subtype,
    const BioAPI_IDENTIFY_POPULATION  *Population,
    uint32_t  TotalNumberOfTemplates,
    BioAPI_BOOL  Binning,
    uint32_t  MaxNumberOfResults,
    uint32_t  *NumberOfResults,
    BioAPI_CANDIDATE  **Candidates,
    int32_t  Timeout,
    BioAPI_BIR_HANDLE  *AuditData);
```

8.4.9.1 Description

This function captures biometric data from the attached device (sensor unit), and compares it against a set of reference BIRs (the *Population)*.

The population that the match takes place against can be presented in one of two ways:

a) in a BIR database identified by an open database handle;

b) input in an array of BIRs;

NOTE: When using a BSP-controlled BIR database, this database must previously have been opened using ***BioAPI_DbOpen***.

There is an option to use an array of BIRs, which can be specified in BioAPI_IDENTIFY_POPULATION_TYPE in the BioAPI_IDENTIFY_POPULATION structure. If it is specified as BioAPI_PRESET_ARRAY_TYPE (3), the array of BIRs which had been previously set in the ***BioAPI_PresetIdentifyPopulation*** call will be used. The preset array of BIRs will be freed internally by the BSP when ***BioAPI_BSPDetach*** is called.

The application shall request a maximum FMR value criterion for a successful match.

The function performs the following actions (in order):

a) captures a sample and processes it as appropriate;

b) determines the set of candidates from the population that match according to the specified criteria;

c) allocates a memory block large enough to contain an array of elements of type BioAPI_CANDIDATE with as many elements as the number of candidates determined in (b);

d) fills the array with the candidate information for all candidates determined in (b), including the *FMRAchieved* of each candidate; and

e) returns the address of the array in the *Candidates* parameter and the size of the array in the *NumberOfResults* parameter.

NOTE: See clause C.4 for information on the use of the FMR concept for normalized scoring and thresholding.

By default, the BSP is responsible for providing the user interface associated with the identify operation. The application may, however, request control of the GUI “look-and-feel” by providing a GUI callback pointer in ***BioAPI_SetGUICallbacks***. See clause C.7 for additional explanation of user interface features.

Since the ***BioAPI_Identify*** operation includes a capture, it serializes use of the sensor device. If two or more biometric applications are racing for the sensor, the losers will wait until the operation completes or the timeout expires. This serialization takes place in all functions that capture data. The BSP is responsible for serializing. It may do this by either returning 'busy' (BioAPI_UNIT_IN_USE) or by queuing requests.

The memory block returned by this BioAPI function call shall be freed by the application using ***BioAPI_Free*** (see clause 8.7.2). Output BIRs can be retrieved by a call to ***BioAPI_GetBIRFromHandle***, which releases the handle, or the handle can be released without retrieving the BIR through ***BioAPI_FreeBIRHandle***.

8.4.9.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

MaxFMRRequested (input) – The requested FMR criterion for successful identification (i.e., the matching threshold).

Subtype (input/optional) – Specifies which subtype (e.g., left/right eye) to capture for identification. A value of BioAPI_NO_SUBTYPE_AVAILABLE (0x00) indicates that the BSP is to select the subtype(s).

NOTE: Not all BSPs support the capture of specific Subtypes.

Population (input) – The population of Reference BIRs (templates) against which the identification is to be performed (by this BSP).

TotalNumberOfTemplates (input) – Specifies the total number of templates stored by the application in the population. A value of NULL is used if not specified.

NOTE: If the total population is distributed over several databases/partitions, then the total size of the population will be greater than the population seen by the BSP. The BSP may map the *FARRequested* to its internal matching threshold based on this total population size.

Binning (input) – A Boolean indicating whether binning is on or off.

NOTE 1: Binning is a search optimization technique that the BSP may employ. It is based on searching a subset of the population according to the intrinsic characteristics of the biometric data. While it may improve the speed of the Match operation, it may also increase the probability of missing a candidate (due to the possibility of mis-binning and as a result, searching a bin which should, but does not, contain the matching BIR).

NOTE 2: Additional information regarding binning is located in clause A.4.6.2.10.

MaxNumberOfResults (input) – Specifies the maximum number of match candidates to be returned as a result of the 1:N match. A value of zero is a request for all candidates.

NumberOfResults (output) – A pointer to the number of candidates returned in the *Candidates* array as a result of the 1:N match.

Candidates (output) – A pointer to the address of an array of elements of type BioAPI_CANDIDATE containing information about the BIRs identified as a result of the match process (i.e., indices associated with BIRs found to meet the match threshold). This list is in rank order, with the best scoring (closest matching) record being first. If no matches are found, no array is allocated and a NULL address is returned. If the *Population* was presented in a BIR database (i.e., the value of BioAPI_IDENTIFY_POPULATION_TYPE is BioAPI_DB_TYPE), the array contains pointers to UUID's corresponding to BIRs stored in the BSP-controlled BIR database. If the *Population* was presented as a passed-in array of BIRs, then BioAPI_IDENTIFY_POPULATION_TYPE has the value BioAPI_ARRAY_TYPE and the array contains pointers to relative indices into the passed-in array.

Timeout (input) – an integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, no array is allocated, and a NULL address is

returned for the *Candidates* parameter. This value can be any positive number. A '-1' value means the BSP's default timeout value will be used.

AuditData (output/optional) – a handle to a BIR containing raw biometric data. This data may be used to provide human-identifiable data of the person at the sensor unit. If the pointer is NULL on input, no audit data is collected. Not all BSPs support the collection of audit data. A BSP may return a handle value of BioAPI_UNSUPPORTED_BIR_HANDLE to indicate *AuditData* is not supported, or a value of BioAPI_INVALID_BIR_HANDLE to indicate that no audit data is available.

8.4.9.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.4.9.4 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_UNABLE_TO_CAPTURE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TIMEOUT_EXPIRED
BioAPIERR_NO_INPUT_BIRS
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_UNIT_IN_USE
BioAPIERR_PRESET_BIR_DOES_NOT_EXIST
BioAPIERR_INVALID_DB_HANDLE
```

See also **BioAPI Error Handling** (clause 11).

NOTE 1: Not all BSPs support 1:N identification. See your BSP programmer's manual to determine if the BSP(s) you are using supports this capability.

NOTE 2: Depending on the BSP and the location and size of the database to be searched, this operation can take a significant amount of time to perform. Check your BSP manual for recommended *Timeout* values.

NOTE 3: The number of match candidates found by the BSP is dependent on the actual FMR of the matching algorithm at the *MaxFMRRequested* threshold setting.

8.4.10 BioAPI_Import

```
BioAPI_RETURN BioAPI BioAPI_Import
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_DATA  *InputData,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *InputFormat,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *OutputFormat,
    BioAPI_BIR_PURPOSE  Purpose,
    BioAPI_BIR_HANDLE  *ConstructedBIR);
```

8.4.10.1 Description

This function passes raw biometric data obtained by a biometric application by any means, and requests the specified BSP attach session to construct a BIR for the purpose specified. *InputData* identifies the memory buffer containing the raw biometric data, while *InputFormat* identifies the form of the raw biometric data. The *InputFormat*s that a particular BSP will be prepared to accept are determined by the BSP (see the error BioAPIERR_UNSUPPORTED_FORMAT). The function returns a handle to the *ConstructedBIR*. If the application needs to acquire the BIR either to store it in a database or to send it to a server, the application can retrieve the data with the ***BioAPI_GetBIRFromHandle*** function, or store it directly using ***BioAPI_DbStoreBIR***.

The output *ConstructedBIR* can be retrieved by a call to ***BioAPI_GetBIRFromHandle***, which releases the handle, or the handle can be released without retrieving the BIR through ***BioAPI_FreeBIRHandle***.

8.4.10.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

InputData (input) – A pointer to image/stream data to import into a Processed BIR. The image/stream conforms to the standard identified by *InputFormat*.

InputFormat (input) – The format of the *InputData*.

OutputFormat (input/optional) – Specifies which BDB format to use for the returned *ConstructedBIR*, if the BSP supports more than one format. A NULL pointer value indicates that the BSP is to select the format.

Purpose (input) – A value indicating the purpose of the *ConstructedBIR*.

ConstructedBIR (output) – A handle to a BIR constructed from the imported biometric data. This BIR may be either an Intermediate or Processed BIR (to be indicated in the header).

8.4.10.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.4.10.4 Errors

```
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_UNABLE_TO_IMPORT
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_PURPOSE_NOT_SUPPORTED
```

See also **BioAPI Error Handling** (clause 11).

8.4.11 BioAPI_PresetIdentifyPopulation

```
BioAPI_RETURN BioAPI BioAPI_PresetIdentifyPopulation
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_IDENTIFY_POPULATION  *Population);
```

8.4.11.1 Description

This function provides the population of BIRs to the BSP, as specified in the *BSPHandle*. The enrollment population to be matched against can be presented in one of two ways:

a) in a BIR database identified by an open database handle;

b) input in an array of BIRs;

The BSP allocates a memory block and transfers the population of BIRs to the memory block with a data format (not standardized) which is supported by the currently attached matching algorithm BioAPI Unit. After this function is called successfully, an application can call ***BioAPI_Identify*** or ***BioAPI_IdentifyMatch,*** specifying BioAPI_PRESET_ARRAY_TYPE in the BioAPI_IDENTIFY_POPULATION structure. The BSP keeps this memory block until another ***BioAPI_PresetIdentifyPopulation*** or ***BioAPI_BSPDetach*** is called.

8.4.11.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

Population (input) – The population of BIRs against which the identification is performed (by this BSP).

8.4.11.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.4.11.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_NO_INPUT_BIRS
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_BIR_NOT_FULLY_PROCESSED
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_FUNCTION_FAILED
```

See also **BioAPI Error Handling** (clause 11).

NOTE: Depending on the size of the database to be transferred and the degree of transformation required, this operation could take a significant amount of time to perform.

8.5 Database Operations

The database operations provide application access to BIR databases on archive BioAPI Units (either directly managed by a BSP or implemented via a BFP).

NOTE 1: See clause C.6 for an explanation of the context for and usage of the database functions.

NOTE 2: Authorization mechanisms to control a biometric application's right to create or delete or write to a database are outside the scope of this part of ISO/IEC 19784 (but may be addressed in 19784-2, *Information technology — Biometric application programming interface — Part 2: Biometric archive function provider interface*).

8.5.1 BioAPI_DbOpen

```
BioAPI_RETURN BioAPI BioAPI_DbOpen
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_UUID  *DbUuid,
    BioAPI_DB_ACCESS_TYPE  AccessRequest,
    BioAPI_DB_HANDLE  *DbHandle,
    BioAPI_DB_MARKER_HANDLE  *MarkerHandle);
```

8.5.1.1 Description

This function opens a BIR database maintained by the currently attached archive of the identified BSP invocation, using the access mode specified by the *AccessRequest*. A new marker is created and set to point to the first record in the BIR database, and a handle for that marker is returned.

Some BSPs may only support a single BIR database or may have a preferred database. The application can allow the BSP to select the BIR database to open by using a NULL pointer value for the database UUID parameter.

NOTE: Every call to this function should be followed at some point by a call to ***BioAPI_DbFreeMarker***, to allow the marker resources to be released. Applications should be aware that every call to ***BioAPI_DbOpen*** results in the allocation of some resources that cannot be automatically freed (otherwise it would be impossible to iterate through the database using the returned marker handle). If the application doesn't want to iterate through the BIR database, it should call ***BioAPI_DbFreeMarker*** immediately, otherwise it should do so after terminating the iteration.

8.5.1.2 Parameters

BSPHandle (input) – The handle of an attached BSP invocation.

DbUuid (input) – A pointer to a UUID identifying the BIR database to be opened. If this value is set to NULL, then the BSP shall select the BIR database to open.

AccessRequest (input) – An indicator of the requested access mode for the BIR database, BioAPI_DB_ACCESS_READ or BioAPI_DB_ACCESS_WRITE. On a single system, only one application at a time is allowed to open a target database in BioAPI_DB_ACCESS_WRITE mode.

DbHandle (output) – A handle for the opened BIR database. The value will be set to BioAPI_DB_INVALID_HANDLE if the function fails.

MarkerHandle (output) – A marker handle that can be subsequently used by the application to iterate through the BIR database from the first record.

8.5.1.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.5.1.4 Errors

```
BioAPIERR_UNABLE_TO_OPEN_DATABASE
BioAPIERR_DATABASE_IS_OPEN
BioAPIERR_DATABASE_IS_LOCKED
BioAPIERR_DATABASE_DOES_NOT_EXIST
BioAPIERR_INVALID_UUID
BioAPIERR_INVALID_ACCESS_REQUEST
BioAPIERR_DATABASE_IS_CORRUPT
```

See also **BioAPI Error Handling** (clause 11).

8.5.2 BioAPI_DbClose

```
BioAPI_RETURN BioAPI BioAPI_DbClose
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_HANDLE  DbHandle);
```

8.5.2.1 Description

This function closes an open BIR database. All markers currently set for records in the database are freed and their handles become invalid.

NOTE: A database opened in BioAPI_DB_ACCESS_WRITE mode can be damaged if it is left unclosed.

8.5.2.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

DbHandle (input) – The DB handle for an open BIR database controlled by the BSP. This specifies the open database to be closed.

8.5.2.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.5.2.4 Errors

```
BioAPIERR_UNABLE_TO_CLOSE_DATABASE
BioAPIERR_INVALID_DB_HANDLE
BioAPIERR_DATABASE_IS_CORRUPT
```

See also **BioAPI Error Handling** (clause 11).

8.5.3 BioAPI_DbCreate

```
BioAPI_RETURN BioAPI BioAPI_DbCreate
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_UUID  *DbUuid,
    uint32_t  NumberOfRecords,
    BioAPI_DB_ACCESS_TYPE  AccessRequest,
    BioAPI_DB_HANDLE  *DbHandle);
```

8.5.3.1 Description

This function creates and opens a new BIR database on the currently attached archive unit of the identified BSP invocation. The identification of the new database is specified by the input parameter *DbUuid* which shall be created by the biometric application, and shall be distinct from any current database UUID supported by that archive unit, whether currently open or not. The newly created BIR database is opened under the specified access mode.

NOTE: This function is used create a new BIR database. It does not transport any information to the archive unit about the new database except its UUID and access conditions. There are archives which are only able to deal with static database sizes or where much higher effort will be needed to manage a database with dynamic size (e.g., smartcards storing templates in transparent or structured files, which might have a static size dependent on the characteristics of the smartcard operating system). Archives may need size information to create a new BIR database. Because the calling application might not have the knowledge about typical or maximum template sizes in bytes, the number of records to be stored at maximum in the database will be provided. Archives which are able to deal with database sizes dynamically may ignore the *NumberOfRecords* parameter.

8.5.3.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

DbUuid (input) – A pointer to a UUID that will identify the BIR database to be created.

NumberOfRecords (input) – The maximum number of records to be stored at in the BIR database.

AccessRequest (input) – An indicator of the requested access mode for the BIR database, such as **read** or **write**. On a single system, only one application at a time is allowed to open a target database in BioAPI_DB_ACCESS_WRITE mode.

DbHandle (output) – The handle to the newly created and open database. The value will be set to BioAPI_DB_INVALID_HANDLE if the function fails.

8.5.3.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.5.3.4 Errors

```
BioAPIERR_UNABLE_TO_CREATE_DATABASE
BioAPIERR_DATABASE_ALREADY_EXISTS
BioAPIERR_INVALID_UUID
BioAPIERR_INVALID_ACCESS_REQUEST
```

See also **BioAPI Error Handling** (clause 11).

8.5.4 BioAPI_DbDelete

```
BioAPI_RETURN BioAPI BioAPI_DbDelete
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_UUID  *DbUuid);
```

8.5.4.1 Description

This function deletes all records from the specified BIR database and removes all state information associated with that database.

8.5.4.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

DbUuid (input) – A pointer to a UUID identifying the BIR database to be deleted.

8.5.4.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.5.4.4 Errors

```
BioAPIERR_DATABASE_IS_OPEN
BioAPIERR_INVALID_UUID
```

See also **BioAPI Error Handling** (clause 11).

8.5.5 BioAPI_DbSetMarker

```
BioAPI_RETURN BioAPI BioAPI_DbSetMarker
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_HANDLE  DbHandle,
    const BioAPI_UUID  *KeyValue,
    BioAPI_DB_MARKER_HANDLE  MarkerHandle);
```

8.5.5.1 Description

The marker identified by the *MarkerHandle* is set to point to the record indicated by the *KeyValue* in the BIR database identified by the *DbHandle*. A NULL value will set the marker to the first record in the database.

NOTE: When an error occurs, the position of the marker does not change.

8.5.5.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

DbHandle (input) – A handle to the open BIR database on the attached archive BioAPI Unit.

KeyValue (input) – The key into the database of the BIR to which the marker is to be set.

MarkerHandle (input) – The handle of the marker that is to be set. This marker handle can be subsequently used by the application to iterate through the BIR database from the new position.

8.5.5.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.5.5.4 Errors

```
BioAPIERR_INVALID_DB_HANDLE
BioAPIERR_RECORD_NOT_FOUND
```

See also **BioAPI Error Handling** (clause 11).

8.5.6 BioAPI_DbFreeMarker

```
BioAPI_RETURN BioAPI BioAPI_DbFreeMarker
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_MARKER_HANDLE  MarkerHandle);
```

8.5.6.1 Description

Frees memory and resources associated with the specified marker and invalidates the *MarkerHandle*.

8.5.6.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

MarkerHandle (input) – The handle of the BIR database marker to be freed.

8.5.6.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.5.6.4 Errors

`BioAPIERR_MARKER_HANDLE_IS_INVALID`

See also **BioAPI Error Handling** (clause 11).

8.5.7 BioAPI_DbStoreBIR

```
BioAPI_RETURN BioAPI BioAPI_DbStoreBIR
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_INPUT_BIR  *BIRToStore,
    BioAPI_DB_HANDLE  DbHandle,
    BioAPI_UUID  *BirUuid);
```

8.5.7.1 Description

The BIR identified by the *BIRToStore* parameter is stored in the open BIR database identified by the *DbHandle* parameter. If the *BIRToStore* is identified by a BIR Handle, the input BIR Handle is freed. If the *BIRToStore* is identified by a database key value, the BIR is retrieved and stored in (copied to) the open database. A new UUID is assigned to the new BIR in the database, and this UUID can be used as a key value to access the BIR later.

8.5.7.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

BIRToStore (input) – The BIR to be stored in the open BIR database (either the BIR, or its handle, or the index to it in another open database).

DbHandle (input) – The handle to the open BIR database.

BirUuid (output) – A UUID that uniquely identifies the new BIR in the BIR database. This UUID cannot be changed. To associate a different BIR with the user, it is necessary to delete the old one, store a new one in the BIR database, and then replace the old UUID with the new one in the application account database. The BIR is added to the tail end of the database.

8.5.7.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.5.7.4 Errors

```
BioAPIERR_INVALID_DB_HANDLE
```

See also **BioAPI Error Handling** (clause 11).

8.5.8 BioAPI_DbGetBIR

```
BioAPI_RETURN BioAPI BioAPI_DbGetBIR
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_HANDLE  DbHandle,
    const BioAPI_UUID  *KeyValue,
    BioAPI_BIR_HANDLE  *RetrievedBIR,
    BioAPI_DB_MARKER_HANDLE  *MarkerHandle);
```

8.5.8.1 Description

The BIR identified by the *KeyValue* parameter in the open BIR database identified by the *DbHandle* parameter is retrieved. The BIR is copied into BSP storage and a handle to it is returned. A marker is created and set to point to the record following the retrieved BIR in the BIR database (or to the first record in the database if the retrieved BIR is the last), and a handle for the marker is returned.

The memory block returned by the BioAPI function call shall be freed by the application using ***BioAPI_Free*** (see clause 8.7.2).

8.5.8.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

DbHandle (input) – The handle to the open BIR database.

KeyValue (input) – The key into the database of the BIR to retrieve.

RetrievedBIR (output) – A handle to the retrieved BIR.

MarkerHandle (output) – A marker handle that can be subsequently used to iterate through the BIR database from the record following the retrieved BIR.

8.5.8.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.5.8.4 Errors

```
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_INVALID_DB_HANDLE
```

See also **BioAPI Error Handling** (clause 11).

8.5.9 BioAPI_DbGetNextBIR

```
BioAPI_RETURN BioAPI BioAPI_DbGetNextBIR
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_HANDLE  DbHandle,
    BioAPI_DB_MARKER_HANDLE  MarkerHandle,
    BioAPI_BIR_HANDLE  *RetrievedBIR,
    BioAPI_UUID  *BirUuid);
```

8.5.9.1 Description

The BIR identified by the *MarkerHandle* parameter is retrieved. The BIR is copied into BSP storage and a handle to it is returned, and a pointer to the UUID that uniquely identifies the BIR in the database is returned. The marker is updated to point to the next record in the database.

NOTE: If there are no more records left in the BIR database, the marker points to an invalid position.

8.5.9.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

DbHandle (input) – The handle to the open BIR database.

MarkerHandle (input/output) – A marker handle indicating which record to retrieve.

RetrievedBIR (output) – A handle to the retrieved BIR.

BirUuid (output) – The UUID that uniquely identifies the retrieved BIR in the BIR database.

8.5.9.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.5.9.4 Errors

```
BioAPIERR_END_OF_DATABASE
BioAPIERR_MARKER_HANDLE_IS_INVALID
BioAPIERR_INVALID_DB_HANDLE
```

See also **BioAPI Error Handling** (clause 11).

8.5.10 BioAPI_DbDeleteBIR

```
BioAPI_RETURN BioAPI BioAPI_DbDeleteBIR
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_HANDLE  DbHandle,
    const BioAPI_UUID  *KeyValue);
```

8.5.10.1 Description

The BIR identified by the *KeyValue* parameter in the open BIR database identified by the *DbHandle* parameter is deleted from the database. If there is a marker set to the deleted BIR, then:

a) if that BIR was not the last BIR in the database, then the marker is moved to the next sequential BIR;

b) otherwise, the marker points to an invalid position. However, its marker handle remains valid and can be used in a subsequent call to *BioAPI_DbSetMarker* to set the marker to point to a different record.

8.5.10.2 Parameters

BSPHandle (input) – The handle of the attached biometric service provider.

DbHandle (input) – The handle to the open BIR database.

KeyValue (input) – The UUID of the BIR to be deleted.

8.5.10.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.5.10.4 Errors

```
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_INVALID_DB_HANDLE
```

See also **BioAPI Error Handling** (clause 11).

8.6 BioAPI Unit operations

8.6.1 BioAPI_SetPowerMode

```
BioAPI_RETURN BioAPI BioAPI_SetPowerMode
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_UNIT_ID  UnitId,
    BioAPI_POWER_MODE  PowerMode);
```

8.6.1.1 Description

This function sets the currently attached BioAPI Unit of the referenced BSP attach session to the requested power mode if the BioAPI Unit supports it.

8.6.1.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

UnitId (input) – The ID of the BioAPI Unit for which the power mode is to be set. The BioAPI_DONT_CARE option is not valid for this function.

PowerMode (input) – A 32-bit value indicating the power mode to set the BioAPI Unit to.

8.6.1.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.6.1.4 Errors

```
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_UNIT_NOT_INSERTED
BioAPIERR_UNIT_IN_USE
```

See also **BioAPI Error Handling** (clause 11).

8.6.2 BioAPI_SetIndicatorStatus

```
BioAPI_RETURN BioAPI BioAPI_SetIndicatorStatus
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_UNIT_ID  UnitId,
    BioAPI_INDICATOR_STATUS  IndicatorStatus);
```

8.6.2.1 Description

This function sets the selected BioAPI Unit to the requested indicator status if the BioAPI Unit supports it. After BioAPI_INDICATOR_ACCEPT or BioAPI_INDICATOR_REJECT is set in the *IndicatorStatus* parameter, the status will not be changed until the application sets another value.

8.6.2.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

UnitId (input) – The ID of the BioAPI Unit for which the indicator status is to be set. The BioAPI_DONT_CARE option is not valid for this function.

IndicatorStatus (input) – A value to which to set the indicator status of the BioAPI Unit.

8.6.2.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.6.2.4 Errors

```
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_UNIT_NOT_INSERTED
BioAPIERR_UNIT_IN_USE
```

See also **BioAPI Error Handling** (clause 11).

8.6.3 BioAPI_GetIndicatorStatus

```
BioAPI_RETURN BioAPI BioAPI_GetIndicatorStatus
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_UNIT_ID  UnitId,
    BioAPI_INDICATOR_STATUS  *IndicatorStatus);
```

8.6.3.1 Description

This function returns the indicator status of the BioAPI Unit if that BioAPI Unit supports it.

8.6.3.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

UnitId (input) – The ID of the BioAPI Unit for which the indicator status is to be obtained. The BioAPI_DONT_CARE option is not valid for this function.

IndicatorStatus (output) – A value for the indicator status of the BioAPI Unit.

8.6.3.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.6.3.4 Errors

```
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_UNIT_NOT_INSERTED
BioAPIERR_UNIT_IN_USE
```

See also **BioAPI Error Handling** (clause 11).

8.6.4 BioAPI_CalibrateSensor

```
BioAPI_RETURN BioAPI BioAPI_CalibrateSensor
       (BioAPI_HANDLE  BSPHandle,
        int32_t  Timeout);
```

8.6.4.1 Description

This function performs a calibration of the attached sensor BioAPI Unit if that sensor unit supports it.

8.6.4.2 Parameters:

BSPHandle (input) – The handle of the attached biometric service provider.

Timeout (input) – An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error. This value can be any positive number. A '-1' value means the BSP's default calibration timeout value will be used.

8.6.4.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.6.4.4 Errors

```
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_UNIT_IN_USE
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_UNIT_NOT_INSERTED
BioAPIERR_CALIBRATION_NOT_SUCCESSFUL
BioAPIERR_TIMEOUT_EXPIRED
```

See also **BioAPI Error Handling** (clause 11).

8.7 Utility Functions

8.7.1 BioAPI_Cancel

```
BioAPI_RETURN BioAPI BioAPI_Cancel
    (BioAPI_HANDLE  BSPHandle);
```

8.7.1.1 Description

This function shall cancel any presently blocked calls associated with *BSPHandle*. The function shall not return until all blocking calls have been cancelled.

8.7.1.2 Parameters

BSPHandle (input) – The handle of the attached BSP.

8.7.1.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.7.1.4 Errors

See **BioAPI Error Handling** (clause 11).

8.7.2 BioAPI_Free

```
BioAPI_RETURN BioAPI BioAPI_Free
    (void*  Ptr);
```

8.7.2.1 Description

This function causes the memory block pointed to by *Ptr* to be deallocated. If *Ptr* is NULL, no action occurs. Otherwise, if *Ptr* does not match a pointer earlier returned by the BioAPI functions, or if the memory block already has been deallocated by a call to ***BioAPI_Free***, the behavior is undefined.

There are some BioSPI functions in which the BSP allocates a memory block which is to be freed by the Framework by calling ***BioSPI_Free***. Whenever the Framework passes one such memory block up to the application, it makes the application responsible for deallocating the memory block. In such cases, the application will simply call ***BioAPI_Free***, and the Framework (on reception of that call) must either free the memory block or call ***BioSPI_Free*** on the appropriate attach session of the appropriate BSP.

NOTE: The Framework may be implemented in such a way that the BSP allocated pointers are not passed to the application. The Framework will in this case move the data returned by the BSP to newly allocated memory, and will call ***BioSPI_Free*** after copying the memory, but before returning back to the application. However, such differences in the behaviour of the Framework are not visible to the application.

In other cases, the Framework itself allocates a memory block and passes it to the application. In such cases, when the application calls ***BioAPI_Free***, the Framework must simply deallocate the memory block, and must not call ***BioSPI_Free*** on any BSP.

This mechanism requires that the Framework keep track of which memory blocks it has allocated itself and which memory blocks it has received from a BSP. For the latter, it must keep track of the BSP and the attach session from which the Framework has received the memory block.

8.7.2.2 Parameters

Ptr (*input*) – A pointer to the memory to free.

8.7.2.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.7.2.4 Errors

See **BioAPI Error Handling** (clause 11).

9 BioAPI Service Provider Interface

9.1 Summary

The service provider interface (SPI) is the programming interface that a BSP shall present in order to interwork with the BioAPI Framework. In general, the SPI is a one-to-one mapping of a function call from a biometric application to the BioAPI Framework (using the BioAPI API specified in clause 8) down to an attach session. The BioAPI Framework routes API calls down to the corresponding SPI of the specified BSP (as identified by the BSP handle parameter in the API function). The following conventions have been adopted in this clause:

a) where an SPI function has parameters that are identical to those of an API function of the same name (apart from the omission of the BSP handle), all that is presented in this clause is the signature of the SPI function, with no further explanation. The reader can then refer to the corresponding API definition for the details;

b) where the parameters of an SPI function differ from those of an API function of the same name, a full definition of the SPI function is presented;

c) Not all API functions have a corresponding SPI function; those functions that do not are handled completely by the BioAPI Framework.

9.2 Type Definitions for Biometric Service Providers

9.2.1 BioSPI_EventHandler

The BioAPI_EventHandler function prototype (clause 7.28) is used to define the event handler interface that enables the BioAPI Framework to receive asynchronous notification of events of type BioAPI_EVENT from a BSP. Example events include insertion or removal of a BioAPI Unit or a fault detection.

The address of a BioSPI_EventHandler function is passed to the BSP during ***BioSPI_BSPLoad***. This is the single event handler that a BSP should call to notify the BioAPI Framework of event types that occur in a loaded BSP.

There is only one event handler for any given BSP that is loaded as a result of a ***BioAPI_BSPLoad*** from one or more applications. It is the responsibility of the BioAPI Framework to notify the (possibly multiple) biometric application event handlers for applications that have loaded the BSP in which an event occurs. Where an insert event has been notified to the Framework before a given biometric application loads the BSP (but after another biometric application has loaded it), the Framework remembers that event and notifies that biometric application's event handler immediately after the load by the biometric application.

The BioAPI Framework forwards events to the biometric application that invoked the corresponding ***BioAPI_BSPLoad*** function. The handler specified in BioSPI_EventHandler can (but need not) be invoked multiple times in response to a single event.

```
typedef BioAPI_RETURN (BioAPI *BioSPI_EventHandler)
       (const BioAPI_UUID *BSPUuid,
       BioAPI_UNIT_ID UnitID,
       const BioAPI_UNIT_SCHEMA *UnitSchema,
       BioAPI_EVENT EventType);
```

9.2.1.1 Definitions

BSPUuid – The UUID of the biometric service provider raising the event.

UnitID – The ID of the BioAPI Unit associated with the event.

UnitSchema – A pointer to the unit schema of the BioAPI Unit associated with the event.

EventType – The BioAPI_EVENT that has occurred.

If the *EventType* is BioAPI_NOTIFY_INSERT, then a unit schema shall be provided (that is, *UnitSchema* shall point to a variable of type BioAPI_UNIT_SCHEMA). Otherwise, *UnitSchema* shall be NULL.

When the framework receives (from a BSP) a call to an event handler that carries a unit schema, the framework shall not call ***BioSPI_Free*** to free the memory block containing the unit schema or a memory block pointed to by any of its members.

9.2.2 BioSPI_BFP_ENUMERATION_HANDLER

This is a callback that the BioAPI Framework exposes to the BSPs to enable them to obtain information about the installed BFPs. This callback is similar to the BioAPI function ***BioAPI_EnumBFPs*** (see clause 8.1.10), but unlike that function, it is not part of the BioAPI API, and thus it is not exposed to applications. Conversely, a BSP can use this callback in order to obtain the same information that an application would obtain by calling the BioAPI function ***BioAPI_EnumBFPs***.

The address of the BioSPI_BFP_ENUMERATION_HANDLER callback is provided by the Framework to a BSP as an input parameter of the ***BioSPI_BSPLoad*** function.

9.2.2.1 Definition

```
typedef BioAPI_RETURN (*BioSPI_BFP_ENUMERATION_HANDLER)
    (BioAPI_BFP_SCHEMA  **BFPSchemaArray,
    uint32_t  *NumberOfElements);
```

This callback provides information about all BFPs currently installed in the component registry. It performs the following actions (in order):

a) allocates a memory block large enough to contain an array of elements of type BioAPI_BFP_SCHEMA with as many elements as the number of installed BFPs;

b) fills the array with the BFP schemas of all installed BFPs; and

c) returns the address of the array in the *BFPSchemaArray* parameter and the number of elements of the array in the *NumberOfElements* parameter.

The memory block containing the array shall be freed by the BSP via a callback to the framework's memory deallocation handler (see clause 9.2.3) when it is no longer needed by the BSP.

The memory blocks pointed to by the *Path* and *BFPProperty* members within each element of the array shall also be freed by the BSP via a callback to the framework's memory deallocation handler (see clause 9.2.3) when they are no longer needed by the application.

9.2.2.2 Parameters

BFPSchemaArray (output) – A pointer to the address of the array of elements of type BioAPI_BFP_SCHEMA (allocated by the framework) containing the BFP schema information.

NumberOfElements (output) – A pointer to the number of elements of the array (which is also the number of BFP schemas in the component registry).

9.2.2.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

9.2.2.4 Errors

See **BioAPI Error Handling** (clause 11).

9.2.3 BioSPI_MEMORY_FREE_HANDLER

This is also called a memory deallocation handler. It is a callback that the BioAPI Framework exposes to the BSPs to enable them to request the deallocation of a memory block that was allocated by the framework to return data to the BSP during a prior callback. Such allocations occur when a BSP calls the framework's BFP enumeration handler (see clause 9.2.2). A memory deallocation handler is similar to the BioAPI function ***BioAPI_Free*** (see clause 8.7.2), but unlike that function, it is not part of the BioAPI API, and thus it is not exposed to applications.

The address of the BioSPI_MEMORY_FREE_HANDLER callback is provided by the Framework to a BSP as an input parameter of the ***BioSPI_BSPLoad*** function.

9.2.3.1 Definition

```
typedef BioAPI_RETURN (*BioSPI_MEMORY_FREE_HANDLER)
    (void* Ptr);
```

This callback causes the memory block pointed to by *Ptr* to be deallocated by the framework. If *Ptr* is NULL, no action occurs. Otherwise, if *Ptr* does not match a pointer earlier returned by a callback, or if the memory block already has been deallocated by an earlier call to the memory deallocation handler, the behavior is undefined.

NOTE: Unlike in the function ***BioAPI_Free***, no requests made to a memory deallocation handler result in a call to ***BioSPI_Free***.

9.2.3.2 Parameters

Ptr (*input*) – A pointer to the memory block to free.

9.2.3.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

9.2.3.4 Errors

See **BioAPI Error Handling** (clause 11).

9.3 Biometric Service Provider Operations

9.3.1 SPI Component Management Operations

9.3.1.1 BioSPI_BSPLoad

```
BioAPI_RETURN BioAPI BioSPI_BSPLoad
    const BioAPI_UUID  *BSPUuid,
    BioSPI_EventHandler  BioAPINotifyCallback,
    BioSPI_BFP_ENUMERATION_HANDLER  BFPEnumerationHandler,
    BioSPI_MEMORY_FREE_HANDLER  MemoryFreeHandler);
```

9.3.1.1.1 Description

This function completes the component initialization process between BioAPI and the biometric service provider. The function ***BioSPI_BSPLoad*** shall not be called more than once without a corresponding call to ***BioSPI_BSPUnload***.

The *BSPUuid* identifies the invoked BSP.

The *BioAPINotifyCallback* defines a callback used to notify the BioAPI Framework of events of type BioAPI_EVENT in any ongoing, attached sessions. The BSP shall retain this information for later use.

The *BFPEnumerationHandler* is the address of the BFP enumeration handler callback provided by the Framework to the BSP. The BSP shall retain this information for later use. The BSP can use the callback whenever it needs to obtain information about the BFPs installed in the Framework.

The *MemoryFreeHandler* is the address of the memory deallocation handler callback provided by the Framework to the BSP. The BSP shall retain this information for later use. The BSP shall use the callback whenever it needs to deallocate a memory block that was allocated by the Framework during a prior callback to the BFP enumeration handler.

NOTE: This is a sister function to ***BioAPI_BSPLoad*** (see clause 8.1.5).

9.3.1.1.2 Parameters

BSPUuid (input) – The UUID of the invoked biometric service provider, provided as a consistency check.

BioAPINotifyCallback (input) – A function pointer for the BioAPI event handler that manages events of type BioAPI_EVENT.

BFPEnumerationHandler (input) – A function pointer to the Framework's BFP enumeration handler that returns BFP schema information for all the installed BFPs.

MemoryFreeHandler (input) – A function pointer to the Framework's memory deallocation handler.

9.3.1.1.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

9.3.1.1.4 Errors

See ***BioAPI_BSPLoad*** (clause 8.1.5).

9.3.1.2 BioSPI_BSPUnload

```
BioAPI_RETURN BioAPI BioSPI_BSPUnload
    (const BioAPI_UUID  *BSPUuid);
```

9.3.1.2.1 Description

This function disables events and de-registers the event-notification function. The biometric service provider may perform cleanup operations, reversing the initialization performed in ***BioSPI_BSPLoad***.

NOTE: This is a sister function to ***BioAPI_BSPUnload*** (see clause 8.1.6).

9.3.1.2.2 Parameters

BSPUuid (input) – The UUID of the invoked biometric service provider.

9.3.1.2.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

9.3.1.2.4 Errors

See ***BioAPI_BSPUnload*** (clause 8.1.6).

9.3.1.3 BioSPI_BSPAttach

```
BioAPI_RETURN BioAPI BioSPI_BSPAttach
    (const BioAPI_UUID  *BSPUuid,
    BioAPI_VERSION  Version,
    const BioAPI_UNIT_LIST_ELEMENT  *UnitList,
    uint32_t  NumUnits,
    BioAPI_HANDLE  BSPHandle);
```

9.3.1.3.1 Description

This function is invoked by the Framework once for each invocation of ***BioAPI_BSPAttach*** specifying the BSP identified by *BSPUuid*.

The biometric service provider shall verify compatibility with the version level specified by *Version*. If the version is not compatible, then this function fails. The BSP should perform all initializations required to support the new BSP invocation.

The BSP shall attach the specified BioAPI Units if they are supported.

NOTE: This is a sister function to ***BioAPI_BSPAttach*** (see clause 8.1.7).

9.3.1.3.2 Parameters

BSPUuid (input) – a pointer to the UUID of the invoked biometric service provider.

Version (input) – The major and minor version number of the BioAPI specification that the application is expecting the BSP to support. The BSP shall determine whether it is compatible with the required version.

UnitList (input) – a pointer to a buffer containing a list of BioAPI_UNIT_LIST_ELEMENT structures indicating to the BSP which BioAPI Units (supported by the BSP) it is to use for this attach session. The structures contain the ID and category of each BioAPI Unit. One of the following will be specified for each category of BioAPI Unit:

a. Selection of a specific BioAPI Unit: The particular BioAPI Unit to be used in this attach session is specified by inclusion of its ID and category.

b. Selection of any BioAPI Unit: When the UnitID is set to BioAPI_DONT_CARE in a particular element, the BSP will choose which BioAPI Unit of that category to use, or will give an error return if it does not support any BioAPI Units of that category. If a particular category is not listed, the BSP will likewise choose a BioAPI Unit of that category to use if it supports a BioAPI Unit of that category (however, there is no error return if it does not).

c. Selection of no BioAPI Unit: When the UnitID is set to BioAPI_DONT_INCLUDE, the BSP will explicitly not attach a BioAPI Unit of the given category, even if it supports one of that category.

NOTE: Any subsequent calls requiring use of a BioAPI Unit of this category will fail with an error return.

NumUnits (input) – The number of BioAPI Unit elements in the list that the pointer *UnitList* is pointing to. If this parameter contains “0”, the BSP selects the BioAPI Unit for all categories of BioAPI Units that the BSP manages directly or indirectly.

BSPHandle (input) – The BioAPI_HANDLE value assigned by the Framework and associated with the attach session being created by this function.

NOTE: Only one BioAPI Unit of each category can be attached for each attach session at any time.

9.3.1.3.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

9.3.1.3.4 Errors

See ***BioAPI_BSPAttach*** (clause 8.1.7).

9.3.1.4 BioSPI_BSPDetach

```
BioAPI_RETURN BioAPI BioSPI_BSPDetach
    (BioAPI_HANDLE  BSPHandle);
```

9.3.1.4.1 Description

This function is invoked by the BioAPI Framework once for each invocation of ***BioAPI_BSPDetach*** specifying the attach session identified by *BSPHandle*. The biometric service provider shall perform all cleanup operations associated with the specified attach handle.

NOTE: This is a sister function to ***BioAPI_BSPDetach*** (see clause 8.1.8).

9.3.1.4.2 Parameters

BSPHandle (input) – The handle associated with the attach session being terminated by this function.

9.3.1.4.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

9.3.1.4.4 Errors

See ***BioAPI_BSPDetach*** (clause 8.1.8).

9.3.1.5 BioSPI_QueryUnits

```
BioAPI_RETURN BioAPI BioSPI_QueryUnits
    (const BioAPI_UUID *Uuid,
    BioAPI_UNIT_SCHEMA  **UnitSchemaArray,
    uint32_t  *NumberOfElements);
```

NOTE: Details of the function definition are located in clause 8.1.9, ***BioAPI_QueryUnits***.

9.3.1.6 BioSPI_QueryBFPs

```
BioAPI_RETURN BioAPI BioSPI_QueryBFPs
    (const BioAPI_UUID  *BSPUuid,
     BioAPI_BFP_LIST_ELEMENT  **BFPList,
     uint32_t  *NumberOfElements);
```

NOTE 1: On an incoming call to this function, the BSP can use the BFP Enumeration Handler callback (see clause 9.2.2) to obtain information about all installed BFPs, and can then create the list of all supported BFPs by checking each entry of the array returned by the callback.

NOTE 2: Details of the function definition are located in clause 8.1.11, ***BioAPI_QueryBFPs***.

9.3.1.7 BioSPI_ControlUnit

```
BioAPI_RETURN BioAPI BioSPI_ControlUnit
    (BioAPI_HANDLE  BSPHandle,
     BioAPI_UNIT_ID  UnitID,
     uint32_t  ControlCode,
     const BioAPI_DATA  *InputData,
     BioAPI_DATA  *OutputData);
```

NOTE: Details of the function definition are located in clause 8.1.12, ***BioAPI_ControlUnit***.

9.3.2 SPI Data Handle Operations

9.3.2.1 BioSPI_FreeBIRHandle

```
BioAPI_RETURN BioAPI BioSPI_FreeBIRHandle
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_BIR_HANDLE  Handle);
```

NOTE: Details of the function definition are located in clause 8.2.1, ***BioAPI_FreeBIRHandle***.

9.3.2.2 BioSPI_GetBIRFromHandle

```
BioAPI_RETURN BioAPI BioSPI_GetBIRFromHandle
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_BIR_HANDLE  Handle,
    BioAPI_BIR  *BIR);
```

NOTE: Details of the function definition are located in clause 8.2.2, ***BioAPI_GetBIRFromHandle***.

9.3.2.3 BioSPI_GetHeaderFromHandle

```
BioAPI_RETURN  BioAPI  BioSPI_GetHeaderFromHandle
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_BIR_HANDLE  Handle,
    BioAPI_BIR_HEADER  *Header);
```

NOTE: Details of the function definition are located in clause 8.2.3, ***BioAPI_GetHeaderFromHandle***.

9.3.3 SPI Callback and Event Operations

9.3.3.1 BioSPI_EnableEvents

```
BioAPI_RETURN BioAPI BioSPI_EnableEvents
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_EVENT_MASK  Events);
```

NOTE: Details of the function definition are located in clause 8.3.1, ***BioAPI_EnableEvents***.

9.3.3.2 BioSPI_SetGUICallbacks

```
BioAPI_RETURN BioAPI BioSPI_SetGUICallbacks
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_GUI_STREAMING_CALLBACK  GuiStreamingCallback,
    void  *GuiStreamingCallbackCtx,
    BioAPI_GUI_STATE_CALLBACK  GuiStateCallback,
    void  *GuiStateCallbackCtx);
```

NOTE: Details of the function definition are located in clause 8.3.2, ***BioAPI_SetGUICallbacks***.

9.3.4 SPI Biometric Operations

9.3.4.1 BioSPI_Capture

```
BioAPI_RETURN BioAPI BioSPI_Capture
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_BIR_PURPOSE  Purpose,
    BioAPI_BIR_SUBTYPE  Subtype,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *OutputFormat,
    BioAPI_BIR_HANDLE  *CapturedBIR,
    int32_t  Timeout,
    BioAPI_BIR_HANDLE  *AuditData);
```

NOTE: Details of the function definition are located in clause 8.4.1, ***BioAPI_Capture***.

9.3.4.2 BioSPI_CreateTemplate

```
BioAPI_RETURN BioAPI BioSPI_CreateTemplate
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_INPUT_BIR  *CapturedBIR,
    const BioAPI_INPUT_BIR  *ReferenceTemplate,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *OutputFormat,
    BioAPI_BIR_HANDLE  *NewTemplate,
    const BioAPI_DATA  *Payload,
    BioAPI_UUID  *TemplateUUID);
```

NOTE: Details of the function definition are located in clause 8.4.2, ***BioAPI_CreateTemplate***.

9.3.4.3 BioSPI_Process

```
BioAPI_RETURN BioAPI BioSPI_Process
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_INPUT_BIR  *CapturedBIR,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *OutputFormat,
    BioAPI_BIR_HANDLE  *ProcessedBIR);
```

NOTE: Details of the function definition are located in clause 8.4.3, ***BioAPI_Process***.

9.3.4.4 BioSPI_ProcessWithAuxBIR

```
BioSPI_RETURN BioAPI BioSPI_ProcessWithAuxBIR
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_INPUT_BIR  *CapturedBIR,
    const BioAPI_INPUT_BIR  *AuxiliaryData,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *OutputFormat,
    BioAPI_BIR_HANDLE  *ProcessedBIR);
```

NOTE: Details of the function definition are located in clause 8.4.4, ***BioAPI_ProcessWithAuxBIR***.

9.3.4.5 BioSPI_VerifyMatch

```
BioAPI_RETURN BioAPI BioSPI_VerifyMatch
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_FMR  MaxFMRRequested,
    const BioAPI_INPUT_BIR  *ProcessedBIR,
    const BioAPI_INPUT_BIR  *ReferenceTemplate,
    BioAPI_BIR_HANDLE  *AdaptedBIR,
    BioAPI_BOOL  *Result,
```

```
    BioAPI_FMR  *FMRAchieved,
    BioAPI_DATA  *Payload);
```

NOTE: Details of the function definition are located in clause 8.4.5, ***BioAPI_VerifyMatch.***

9.3.4.6 BioSPI_IdentifyMatch

```
BioAPI_RETURN BioAPI BioSPI_IdentifyMatch
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_FMR  MaxFMRRequested,
    const BioAPI_INPUT_BIR  *ProcessedBIR,
    const BioAPI_IDENTIFY_POPULATION  *Population,
    uint32_t  TotalNumberOfTemplates,
    BioAPI_BOOL  Binning,
    uint32_t  MaxNumberOfResults,
    uint32_t  *NumberOfResults,
    BioAPI_CANDIDATE  **Candidates,
    int32_t  Timeout);
```

NOTE: Details of the function definition are located in clause 8.4.6, ***BioAPI_IdentifyMatch***.

9.3.4.7 BioSPI_Enroll

```
BioAPI_RETURN BioAPI BioSPI_Enroll
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_BIR_PURPOSE  Purpose,
    BioAPI_BIR_SUBTYPE  Subtype,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *OutputFormat,
    const BioAPI_INPUT_BIR  *ReferenceTemplate,
    BioAPI_BIR_HANDLE  *NewTemplate,
    const BioAPI_DATA  *Payload,
    int32_t  Timeout,
    BioAPI_BIR_HANDLE  *AuditData,
    BioAPI_UUID  *TemplateUUID);
```

NOTE: Details of the function definition are located in clause 8.4.7, ***BioAPI_Enroll***.

9.3.4.8 BioSPI_Verify

```
BioAPI_RETURN BioAPI BioSPI_Verify
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_FMR  MaxFMRRequested,
    const BioAPI_INPUT_BIR  *ReferenceTemplate,
    BioAPI_BIR_SUBTYPE  Subtype,
    BioAPI_BIR_HANDLE  *AdaptedBIR,
    BioAPI_BOOL  *Result,
    BioAPI_FMR  *FMRAchieved,
    BioAPI_DATA  *Payload,
    int32_t  Timeout,
    BioAPI_BIR_HANDLE  *AuditData);
```

NOTE: Details of the function definition are located in clause 8.4.8, ***BioAPI_Verify***.

9.3.4.9 BioSPI_Identify

```
BioAPI_RETURN BioAPI BioSPI_Identify
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_FMR  MaxFMRRequested,
    BioAPI_BIR_SUBTYPE  Subtype,
    const BioAPI_IDENTIFY_POPULATION  *Population,
```

```
    uint32_t  TotalNumberOfTemplates,
    BioAPI_BOOL  Binning,
    uint32_t  MaxNumberOfResults,
    uint32_t  *NumberOfResults,
    BioAPI_CANDIDATE  **Candidates,
    int32_t  Timeout,
    BioAPI_BIR_HANDLE  *AuditData);
```

NOTE: Details of the function definition are located in clause 8.4.9, ***BioAPI_Identify.***

9.3.4.10 BioSPI_Import

```
BioAPI_RETURN BioAPI BioSPI_Import
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_DATA  *InputData,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *InputFormat,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT  *OutputFormat,
    BioAPI_BIR_PURPOSE  Purpose,
    BioAPI_BIR_HANDLE  *ConstructedBIR);
```

NOTE: Details of the function definition are located in clause 8.4.10, ***BioAPI_Import.***

9.3.4.11 BioSPI_PresetIdentifyPopulation

```
BioAPI_RETURN BioAPI BioSPI_PresetIdentifyPopulation
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_IDENTIFY_POPULATION  *Population);
```

NOTE: Details of the function definition are located in clause 8.4.11, ***BioAPI_PresetIdentifyPopulation.***

9.3.5 SPI Database Operations

9.3.5.1 BioSPI_DbOpen

```
BioAPI_RETURN BioAPI BioSPI_DbOpen
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_UUID *DbUuid,
    BioAPI_DB_ACCESS_TYPE  AccessRequest,
    BioAPI_DB_HANDLE  *DbHandle,
    BioAPI_DB_MARKER_HANDLE  *MarkerHandle);
```

NOTE: Details of the function definition are located in clause 8.5.1, ***BioAPI_DbOpen***.

9.3.5.2 BioSPI_DbClose

```
BioAPI_RETURN BioAPI BioSPI_DbClose
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_HANDLE  DbHandle);
```

NOTE: Details of the function definition are located in clause 8.5.2, ***BioAPI_DbClose.***

9.3.5.3 BioSPI_DbCreate

```
BioAPI_RETURN BioAPI BioSPI_DbCreate
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_UUID  *DbUuid,
    uint32_t  NumberOfRecords,
    BioAPI_DB_ACCESS_TYPE  AccessRequest,
    BioAPI_DB_HANDLE  *DbHandle);
```

NOTE: Details of the function definition are located in clause 8.5.3, ***BioAPI_DbCreate.***

9.3.5.4 BioSPI_DbDelete

```
BioAPI_RETURN BioAPI BioSPI_DbDelete
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_UUID  *DbUuid);
```

NOTE: Details of the function definition are located in clause 8.5.4, ***BioAPI_DbDelete***.

9.3.5.5 BioSPI_DbSetMarker

```
BioAPI_RETURN BioAPI BioSPI_DbSetMarker
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_HANDLE  DbHandle,
    const BioAPI_UUID  *KeyValue,
    BioAPI_DB_MARKER_HANDLE  MarkerHandle);
```

NOTE: Details of the function definition are located in clause 8.5.5, ***BioAPI_DbSetMarker.***

9.3.5.6 BioSPI_DbFreeMarker

```
BioAPI_RETURN BioAPI BioSPI_DbFreeMarker
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_MARKER_HANDLE  MarkerHandle);
```

NOTE: Details of the function definition are located in clause 8.5.6, ***BioAPI_DbFreeMarker.***

9.3.5.7 BioSPI_DbStoreBIR

```
BioAPI_RETURN BioAPI BioSPI_DbStoreBIR
    (BioAPI_HANDLE  BSPHandle,
    const BioAPI_INPUT_BIR  *BIRToStore,
    BioAPI_DB_HANDLE  DbHandle,
    BioAPI_UUID  *BirUuid);
```

NOTE: Details of the function definition are located in clause 8.5.7, ***BioAPI_DbStoreBIR.***

9.3.5.8 BioSPI_DbGetBIR

```
BioAPI_RETURN BioAPI BioSPI_DbGetBIR
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_HANDLE  DbHandle,
    const BioAPI_UUID  *KeyValue,
    BioAPI_BIR_HANDLE  *RetrievedBIR,
    BioAPI_DB_MARKER_HANDLE  *MarkerHandle);
```

NOTE: Details of the function definition are located in clause 8.5.8, ***BioAPI_DbGetBIR.***

9.3.5.9 BioSPI_DbGetNextBIR

```
BioAPI_RETURN BioAPI BioSPI_DbGetNextBIR
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_HANDLE  DbHandle,
    BioAPI_DB_MARKER_HANDLE  MarkerHandle,
    BioAPI_BIR_HANDLE  *RetrievedBIR,
    BioAPI_UUID  *BirUuid);
```

NOTE: Details of the function definition are located in clause 8.5.9, ***BioAPI_DbGetNextBIR.***

9.3.5.10 BioSPI_DbDeleteBIR

```
BioAPI_RETURN BioAPI BioSPI_DbDeleteBIR
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_DB_HANDLE  DbHandle,
    const BioAPI_UUID  *KeyValue);
```

NOTE: Details of the function definition are located in clause 8.5.10, ***BioAPI_DbDeleteBIR.***

9.3.6 SPI BioAPI Unit operations

9.3.6.1 BioSPI_SetPowerMode

```
BioAPI_RETURN BioAPI BioSPI_SetPowerMode
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_UNIT_ID  UnitId,
    BioAPI_POWER_MODE  PowerMode);
```

NOTE: Details of the function definition are located in clause 8.6.1, ***BioAPI_SetPowerMode.***

9.3.6.2 BioSPI_SetIndicatorStatus

```
BioAPI_RETURN BioAPI BioSPI_SetIndicatorStatus
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_UNIT_ID  UnitId,
    BioAPI_INDICATOR_STATUS  IndicatorStatus);
```

NOTE: Details of the function definition are located in clause 8.6.2, ***BioAPI_SetIndicatorStatus.***

9.3.6.3 BioSPI_GetIndicatorStatus

```
BioAPI_RETURN BioAPI BioSPI_GetIndicatorStatus
    (BioAPI_HANDLE  BSPHandle,
    BioAPI_UNIT_ID  UnitId,
    BioAPI_INDICATOR_STATUS  *IndicatorStatus);
```

NOTE: Details of the function definition are located in clause 8.6.3, ***BioAPI_GetIndicatorStatus.***

9.3.6.4 BioSPI_CalibrateSensor

```
BioAPI_RETURN BioAPI BioSPI_CalibrateSensor
       (BioAPI_HANDLE  BSPHandle,
        int32_t  Timeout);
```

NOTE: Details of the function definition are located in clause 8.6.4, ***BioAPI_CalibrateSensor.***

9.3.7 SPI Utility Functions

9.3.7.1 BioSPI_Cancel

```
BioAPI_RETURN BioAPI BioSPI_Cancel
    (BioAPI_HANDLE  BSPHandle);
```

NOTE: Details of the function definition are located in clause 8.7.1, ***BioAPI_Cancel.***

9.3.7.2 BioSPI_Free

```
BioAPI_RETURN BioAPI BioSPI_Free
    (void*  Ptr);
```

NOTE: Details of the function definition are located in clause 8.7.2, ***BioAPI_Free***.

10 Component registry interface

The component registry provides a registry for static information describing the capabilities of BioAPI components. The component registry may be queried by biometric applications to obtain this information. The component registry is part of the BioAPI Framework.

An application (such as an installation wizard) that installs a BioAPI component (BioAPI Framework, BSP, or BFP) posts information about that component into the component registry. This information is used by a biometric application to determine what BSPs and BFPs have been installed. The biometric application can use this information dynamically in its decision logic. For example, it can decide whether or not to make a specific (optional) function call to a given BSP depending on whether the registry entry for that BSP indicates that the call is supported. Additionally, the component registry provides information regarding the BDB formats supported by a BSP (see clause 6.7 and Annex B) and default values of a BSP for various common parameters (such as timeouts).

NOTE: Information regarding BioAPI Units is not stored in the component registry but is obtained upon loading a BSP (as an insert event) or in response to a query (***BioAPI_QueryUnits***).

The component registry is designed to be platform independent and does not necessarily (but could) use the built-in registry of any specific operating system (such as the Microsoft Windows™ registry).

The means of posting information to the component registry is defined in clause 10.2.

10.1 BioAPI Registry Schema

The various components of a BioAPI-based biometric system are represented by records in the component registry. These records are operating system independent. The following sub-clauses define the information that is held in the registry for each component of a BioAPI-based biometric system.

NOTE: All string types defined as elements of the component schemas are character encoded as specified by ISO/IEC 10646 (UTF-8 transformation) unless otherwise specified in the data structures of clause 7.

10.1.1 Framework Schema

This schema contains attributes of the BioAPI Framework itself.

Table 2 — Framework Schema Elements

Field Name	Field Data Type	Description
FrameworkUUID	UINT8_T [16]	UUID uniquely identifying the BioAPI Framework.
Description	STRING	The BioAPI Framework product description, in text.
Path	STRING	Path (including filename) of the executable code of the BioAPI Framework
Spec Version	UINT8_T	The BioAPI Specification Version that is supported
Product Version	STRING	The BioAPI Framework product version
Vendor	STRING	The BioAPI Framework vendor name, in text.
FW Property ID	UINT8_T [16]	UUID identifying the format of the Framework Properties element (assigned by the vendor)
FW Property	DATA	Vendor specified information about the Framework

NOTE: See also clause 7.30.

10.1.2 BSP Schema

This schema describes the capabilities of a BSP. There is one BSP Schema for each BSP that has been installed on the biometric system.

Table 3 — BSP Schema Elements

Field Name	Field Data Type	Description
BSPUUID	UINT8_T [16]	UUID uniquely identifying the BSP
Description	STRING	Text descriptive name of the BSP
Path	STRING	Path where the BSP executable is located, including the filename for the executable code of the BSP (may be a URL)
Spec Version	UINT8_T	The BioAPI Specification version that is supported
Product Version	STRING	The BSP product version
Vendor	STRING	The BSP vendor name, in text
BSP Supported Formats	MULTIUINT32_T	An array of 2-byte integer pairs. Each pair specifying a supported BDB format. See BioAPI_BIR_BIOMETRIC_DATA_FORMAT
Number of Supported Formats	UINT32_T	Integer specifying the number of BDB formats supported by the BSP (Number of 2-byte integer pairs in BSP Supported Formats)
Factors Mask	UINT32_T	A mask which indicates what forms of authentication are supported by the BSP (See BioAPI_BIR_BIOMETRIC_TYPE)
Operations	UINT32_T	Operations (functions) supported by the BSP (See BioAPI_OPERATIONS_MASK)
Options	UINT32_T	Options supported by the BSP (See BioAPI_OPTIONS_MASK)
Payload Policy	UINT32_T	Threshold setting (maximum FMR value) used to determine when to release a payload (See BioAPI_FMR)
Max Payload Size	UINT32_T	Maximum size in bytes of a payload
Default Verify Timeout	INT32_T	Default timeout value (in milliseconds) used by the BSP for verify operations when no timeout is set by the biometric application
Default Identify Timeout	INT32_T	Default timeout value (in milliseconds) used by the BSP for identify operations when no timeout is set by the biometric application.
Default Capture Timeout	INT32_T	Default timeout value (in milliseconds) used by the BSP for capture operations when no timeout is set by the biometric application.
Default Enroll Timeout	INT32_T	Default timeout value (in milliseconds) used by the BSP for enroll operations when no timeout is set by the biometric application.
Default Calibrate Timeout	INT32_T	Default timeout value in milliseconds used by the BSP for sensor calibration operations when no timeout is specified by the application.
MAX BSP DB size	UINT32_T	Maximum size of a BIR database. If NULL, no BSP database exists for this BSP.[1]
Max Identify Population	UINT32_T	Largest population supported by the Identify function.[1] Unlimited = 0xFFFFFFFF.

1 Applies only when a BSP is only capable of directly managing a single archive unit. Value=0 means it is capable of managing multiple units (either directly or through a BFP interface) & information about these units will be provided as part of the insert notification (part of Unit Schema).

NOTE: See also clause 7.16.

10.1.3 BFP Schema

This schema describes the capabilities of a BFP. There is one BFP Schema for each BFP that has been installed on the biometric system.

Table 4 — BFP Schema Elements

Field Name	Field Data Type	Description
BFP UUID	UINT8_T [16]	UUID uniquely identifying the BFP
BFP Category	UINT32_T	Category of the BFP identified by the BFP UUID
Description	STRING	Text descriptive name of the BFP
Path	STRING	Path where the BFP executable is located, including the filename for the executable code of the BFP (may be a URL)
Spec Version	UINT8_T	The BioAPI FPI Specification version supported[1]
Product Version	STRING	The BFP product version
Vendor	STRING	The BFP vendor name, in text
BFP Supported Formats	MULTIUINT32_T	An array of 2-byte integer pairs. Each pair specifying a supported BDB format. See BioAPI_BIR_BIOMETRIC_DATA_FORMAT
Number of Supported Formats	UINT32_T	Integer specifying the number of BDB formats supported by the BFP (Number of 2-byte integer pairs in BFP Supported Formats)
Factors Mask	UINT32_T	A mask which indicates what forms of authentication are supported by the BFP (See BioAPI_BIR_BIOMETRIC_TYPE)
BFP Property ID	UINT8_T [16]	UUID identifying the format of the BFP Properties element (assigned by the vendor)
BFP Property	DATA	Vendor specified information about the BFP
1 Function Provider Interface (FPI) specifications will be published as separate parts of ISO/IEC 19784.		

NOTE: See also clause 7.3.

10.2 Component registry functions

The following component registry utility functions are provided to ease the development of BioAPI compatible applications and BSPs.

Installation functions are provided so that a BSP or BFP implementer can populate the BSP or BFP schema within the component registry. The component registry is independent of the operating system.

10.2.1 BioAPI_Util_InstallBSP

10.2.1.1 Description

This function installs, updates (refreshes), or removes the references to a BSP in the component registry.

```
BioAPI_RETURN BioAPI_Util_InstallBSP (
    BioAPI_INSTALL_ACTION  Action,
    BioAPI_INSTALL_ERROR  *Error,
    const BioAPI_BSP_SCHEMA  *BSPSchema);
```

Upon initiation of this function for an install, the BioAPI Framework shall create the BSP schema entry in the component registry and populate it with the content of the input BSP schema.

Upon initiation of this function for a refresh, the BioAPI Framework shall replace the existing entries for the BSP schema in the component registry with the content of the input BSP schema, based on the UUID of the BSP within that schema.

Upon initiation of this function for a deinstallation, the BioAPI Framework shall remove the BSP schema entry in the component registry for the BSP indicated by the *BSPUuid* within the input schema.

10.2.1.2 Parameters

Action (input) – Installation action to be performed. Valid values are:

Value	Description
BioAPI_INSTALL_ACTION_INSTALL	Install BSP (add entry)
BioAPI_INSTALL_ACTION_REFRESH	Refresh the information in the component registry.
BioAPI_INSTALL_ACTION_UNINSTALL	Uninstall the BSP (remove entry)

Error (output) – A pointer to a BioAPI_INSTALL_ERROR structure. If the function fails, the structure contains additional information regarding the error state.

BSPSchema (input) – A pointer to elements of the BSP schema (defined in 7.16) describing the features and characteristics of the BSP to be installed. When the function is called with *Action* set to BioAPI_INSTALL_ACTION_UNINSTALL, it is only necessary to set the *BSPUuid* element of the *BSPSchema* parameter.

10.2.1.3 Return Values

If the function is successful, it returns BioAPI_OK. If the function fails, it returns a valid BioAPI error code (see clause 11). The parameter *Error* contains additional error information.

10.2.2 BioAPI_Util_InstallBFP

10.2.2.1 Description

This function installs, updates (refreshes), or removes the entries for a BioAPI BFP in the component registry.

```
BioAPI_RETURN BioAPI_Util_InstallBFP (
    BioAPI_INSTALL_ACTION  Action,
    BioAPI_INSTALL_ERROR  *Error,
    const BioAPI_BFP_SCHEMA  *BFPSchema);
```

10.2.2.2 Parameters

Action (input) – Installation action to be performed. Valid values are:

Value	Description
BioAPI_INSTALL_ACTION_INSTALL	Install BFP (add entry)
BioAPI_INSTALL_ACTION_REFRESH	Refresh the information in the component registry.
BioAPI_INSTALL_ACTION_UNINSTALL	Uninstall the BFP (remove entry)

Error (output) – A pointer to a BioAPI_INSTALL_ERROR structure. If the function fails, the structure contains additional information regarding the error state.

BFPSchema (input/output) – A pointer to component registry elements describing the features and characteristics of the BFP to be installed (see clause 7.3). When the function is called with *Action* set to BioAPI_INSTALL_ACTION_UNINSTALL, it is only necessary to set the *BFPUuid* element of the BFP*Schema* parameter.

10.2.2.3 Return Values

If the function is successful, it returns BioAPI_OK. If the function fails, it returns a valid BioAPI error code (see clause 11). The parameter *Error* contains additional error information.

11 BioAPI error handling

All BioAPI functions return a value of type BioAPI_RETURN. The value BioAPI_OK indicates that the function was successful. Any other value is an error value. Allowance has been made for returning error values originated from the BioAPI Framework and also error values that originate from a BSP or from a BioAPI Unit attached to a BSP.

11.1 Error Values and Error Codes Scheme

Error Value refers to the entire 32-bit BioAPI_RETURN value.

Error Code refers to the low-order 24 bits of the Error Value, and identifies the actual error situation.

The 8 high-order bits of the Error Value identify the source of the error, as follows:

a) The BioAPI Framework

b) A BSP

c) A BioAPI Unit attached to a BSP

11.2 Error Codes and Error Value Enumeration

11.2.1 BioAPI Error Value Constants

```
#define BIOAPI_FRAMEWORK_ERROR        (0x00000000)
#define BIOAPI_BSP_ERROR              (0x01000000)
#define BIOAPI_UNIT_ERROR             (0x02000000)
```

11.2.2 Implementation-Specific Error Codes

The error codes 0x000000 – 0x0000ff are reserved for implementation-specific use.

11.2.3 General Error Codes

```
#define BioAPIERR_INTERNAL_ERROR (0x000101)
```

General system error; indicates that an operating system or internal state error has occurred and the system may not be in a known state.

```
#define BioAPIERR_MEMORY_ERROR (0x000102)
```

A memory error occurred.

```
#define BioAPIERR_INVALID_POINTER (0x000103)
```

An input/output function parameter or input/output field inside of a data structure is an invalid pointer.

```
#define BioAPIERR_INVALID_INPUT_POINTER (0x000104)
```

An input function parameter or input field in a data structure is an invalid pointer.

```
#define BioAPIERR_INVALID_OUTPUT_POINTER (0x000105)
```

An output function parameter or output field in a data structure is an invalid pointer.

```
#define BioAPIERR_FUNCTION_NOT_SUPPORTED (0x000106)
```

The function is not implemented by the biometric service provider.

```
#define BioAPIERR_OS_ACCESS_DENIED (0x000107)
```

The operating system denied access to a required resource.

```
#define BioAPIERR_FUNCTION_FAILED (0x000108)
```

The function failed for an unknown reason.

```
#define BioAPIERR_INVALID_UUID (0x000109)
```

An input UUID is invalid.

(May occur if a component requested by this UUID is not present on the system or cannot be found.)

```
#define BioAPIERR_INCOMPATIBLE_VERSION (0x00010a)
```

Version incompatibility.

(May occur if the called component cannot support the version of the BioAPI specification expected by the application.)

```
#define BioAPIERR_INVALID_DATA (0x00010b)
```

The data in an input parameter is invalid.

```
#define BioAPIERR_UNABLE_TO_CAPTURE (0x00010c)
```

The associated BSP is unable to capture raw samples from the requested BioAPI Unit.

```
#define BioAPIERR_TOO_MANY_HANDLES (0x00010d)
```

The associated BSP has no more space to allocate BIR handles.

```
#define BioAPIERR_TIMEOUT_EXPIRED (0x00010e)
```

The function has been terminated because the timeout value has expired.

```
#define BioAPIERR_INVALID_BIR (0x00010f)
```

The input BIR is invalid for the purpose required.

```
#define BioAPIERR_BIR_SIGNATURE_FAILURE (0x000110)
```

The associated BSP could not validate the signature on the BIR.

```
#define BioAPIERR_UNABLE_TO_STORE_PAYLOAD (0x000111)
```

The associated BSP is unable to store the payload.

```
define BioAPIERR_NO_INPUT_BIRS (0x000112)
```

The identify population is NULL.

```
#define BioAPIERR_UNSUPPORTED_FORMAT (0x000113)
```

The associated BSP does not support the BDB format requested.

```
#define BioAPIERR_UNABLE_TO_IMPORT (0x000114)
```

The associated BSP was unable to construct a BIR from the input data.

```
#define BioAPIERR_INCONSISTENT_PURPOSE (0x000115)
```

The purpose recorded in the BIR and/or the requested purpose are inconsistent with the function being performed.

```
#define BioAPIERR_BIR_NOT_FULLY_PROCESSED (0x000116)
```

The function requires a fully processed BIR.

```
#define BioAPIERR_PURPOSE_NOT_SUPPORTED (0x000117)
```

The BSP does not support the requested purpose.

```
#define BioAPIERR_USER_CANCELLED (0x000118)
```

User cancelled operation before completion or timeout.

```
#define BioAPIERR_UNIT_IN_USE (0x000119)
```

BSP (or BioAPI Unit attached to BSP) is currently being used by another biometric application.

```
#define BioAPIERR_INVALID_BSP_HANDLE (0x00011a)
```

The given BSP handle is not valid.

```
#define BioAPIERR_FRAMEWORK_NOT_INITIALIZED (0x00011b)
```

A function has been called without initializing the BioAPI Framework.

```
#define BioAPIERR_INVALID_BIR_HANDLE (0x00011c)
```

BIR handle is invalid (does not exist or has been released).

```
#define BioAPIERR_CALIBRATION_NOT_SUCCESSFUL (0x00011d)
```

The attempted calibration of a sensor unit was not able to be successfully completed.

```
#define BioAPIERR_PRESET_BIR_DOES_NOT_EXIST (0x00011e)
```

No preset BIR population has been established.

```
#define BioAPIERR_BIR_DECRYPTION_FAILURE (0x00011f)
```

The BSP could not decrypt an input BIR (and thus was unable to use it for the requested operation).

11.2.4 Component Management Error Codes

```
#define BioAPIERR_COMPONENT_FILE_REF_NOT_FOUND (0x000201)
```

A reference to the file for the component being loaded cannot be found.

```
#define BioAPIERR_BSP_LOAD_FAIL (0x000202)
```

Framework was unable to successfully load the BSP.

```
#define BioAPIERR_BSP_NOT_LOADED (0x000203)
```

BSP for which an action was requested is not loaded.

```
#define BioAPIERR_UNIT_NOT_INSERTED (0x000204)
```

BioAPI Unit for which an action was requested is not in the inserted state.

```
#define BioAPIERR_INVALID_UNIT_ID (0x000205)
```

An invalid BioAPI Unit ID was requested.

```
#define BioAPIERR_INVALID_CATEGORY (0x000206)
```

An invalid category of BFP or BioAPI Unit was requested.

11.2.5 Database Error Values

```
#define BioAPIERR_INVALID_DB_HANDLE (0x000300)
```

Invalid database handle.

```
#define BioAPIERR_UNABLE_TO_OPEN_DATABASE (0x000301)
```

The associated BSP is unable to open the specified database.

```
#define BioAPIERR_DATABASE_IS_LOCKED (0x000302)
```

The database cannot be opened for the access requested because it is locked.

```
#define BioAPIERR_DATABASE_DOES_NOT_EXIST (0x000303)
```

The specified database does not exist.

```
#define BioAPIERR_DATABASE_ALREADY_EXISTS (0x000304)
```

Create failed because the database already exists.

```
#define BioAPIERR_INVALID_DATABASE_NAME (0x000305)
```

Invalid database name (UUID).

```
#define BioAPIERR_RECORD_NOT_FOUND (0x000306)
```

No record exists with the requested key.

```
#define BioAPIERR_MARKER_HANDLE_IS_INVALID (0x000307)
```

The specified marker handle is invalid.

```
#define BioAPIERR_DATABASE_IS_OPEN (0x000308)
```

The database is already open.

```
#define BioAPIERR_INVALID_ACCESS_REQUEST (0x000309)
```

Unrecognized access type.

```
#define BioAPIERR_END_OF_DATABASE (0x00030a)
```

End of database has been reached.

```
#define BioAPIERR_UNABLE_TO_CREATE_DATABASE (0x00030b)
```

The associated BSP cannot create the database.

```
#define BioAPIERR_UNABLE_TO_CLOSE_DATABASE (0x00030c)
```

The associated BSP cannot close the database.

```
#define BioAPIERR_UNABLE_TO_DELETE_DATABASE (0x00030d)
```

The associated BSP cannot delete the database.

```
#define BioAPIERR_DATABASE_IS_CORRUPT (0x00030e)
```

The specified database is corrupt.

11.2.6 Location Error Values

In various biometric technologies like fingerprint, face, iris, retina or others, biometric devices (sensor units) have three dimensions from the user's viewpoint. The error codes in this clause describe the errors that can be caused by improper placement of a user's biometric feature on the biometric device when attempting to capture their biometric sample. Figure 4 below shows an example of a fingerprint sensor, which can be mounted vertically (Case a) or horizontally (Case b). When the placement of the biometric feature is improper, error codes are required to provide appropriate user feedback. If the sensor is placed vertically, the words 'forward' and 'backward' will be the expressions of intensity of the contact or distance, and if placed horizontally, these words will be the expressions of lengthwise distances.

If the BSP or BioAPI Unit generating the error code knows whether the biometric device is oriented vertically or horizontally, it can then choose an error code that describes the user's situation appropriately. If it does not, then a more general error code has to be used.

NOTE: These errors can be returned from ***BioAPI_CreateTemplate***, ***BioAPI_Process***, ***BioAPI_ProcessWithAuxBIR***, ***BioAPI_VerifyMatch***, and ***BioAPI_IdentifyMatch***. The BSP controlled GUI in the other biometric operations (***BioAPI_Capture***, ***BioAPI_Enroll***, ***BioAPI_Verify***, and ***BioAPI_Identify***) will instruct the user to capture a correct sample.

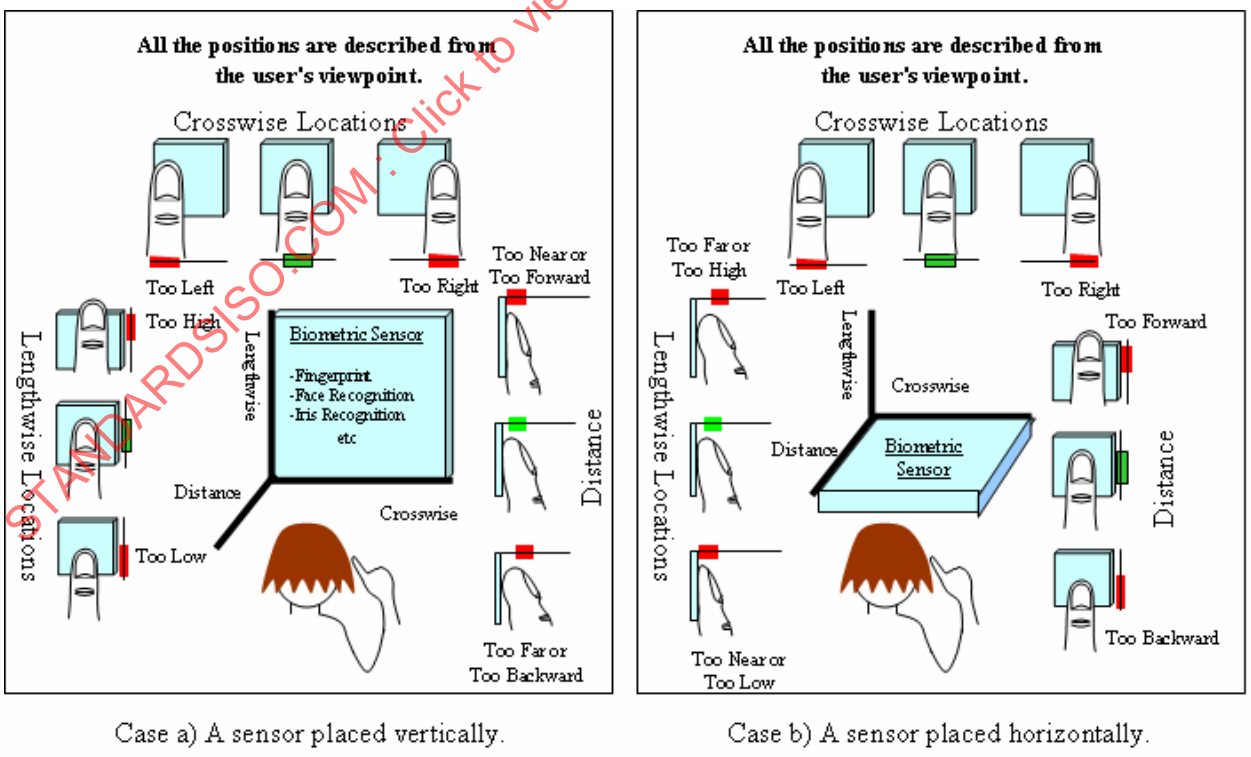

Figure 4 — Biometric Sample Locations (Example of a fingerprint sensor)

Biometric applications can use these error codes for user feedback to obtain better samples regardless of what kind of biometric technology they are using, as long as it has three dimensions. The error codes are specified below.

If the BSP or BioAPI Unit generating the error code is not certain about the orientation of the sensor, it can return generic error codes.

11.2.6.1 General location error codes

```
#define BioAPIERR_LOCATION_ERROR (0x000400)
```

A general location error.

```
#define BioAPIERR_OUT_OF_FRAME (0x000401)
```

Invalid horizontal or vertical position.

```
#define BioAPIERR_INVALID_CROSSWISE_POSITION (0x000402)
```

Invalid crosswise position.

```
#define BioAPIERR_INVALID_LENGTHWISE_POSITION (0x000403)
```

Invalid lengthwise position.

```
#define BioAPIERR_INVALID_DISTANCE (0x000404)
```

Invalid distance.

11.2.6.2 Specific location error codes

```
#define BioAPIERR_LOCATION_TOO_RIGHT (0x000405)
```

The position was too far to the right.

```
#define BioAPIERR_LOCATION_TOO_LEFT (0x000406)
```

The position was too far to the left.

```
#define BioAPIERR_LOCATION_TOO_HIGH (0x000407)
```

The position was too high.

```
#define BioAPIERR_LOCATION_TOO_LOW (0x000408)
```

The position was too low.

```
#define BioAPIERR_LOCATION_TOO_FAR (0x000409)
```

The position was too far away.

```
#define BioAPIERR_LOCATION_TOO_NEAR (0x00040a)
```

The position was too near (close).

```
#define BioAPIERR_LOCATION_TOO_FORWARD (0x00040b)
```

The position was too far forward.

```
#define BioAPIERR_LOCATION_TOO_BACKWARD (0x00040c)
```

The position was too far backward.

11.2.7 Quality Error Codes

```
#define BioAPIERR_QUALITY_ERROR (0x000501)
```

Sample quality is too poor for the operation to succeed.

NOTE: Quality errors can be returned from any function which receives a BioAPI BIR input.

Annex A
(normative)

Conformance

A.1 General

A.1.1 Conformance to this part of ISO/IEC 19784 falls into the following three classes:

- BioAPI conformant biometric application
- BioAPI conformant BioAPI Framework
- BioAPI conformant BSP, comprising one of the following sub-classes:
    - BioAPI conformant Verification BSP
    - BioAPI conformant Identification BSP
    - BioAPI conformant Capture BSP
    - BioAPI conformant Verification Engine
    - BioAPI conformant Identification Engine

A.1.2 Conformance requirements for biometric applications, BioAPI Frameworks, and for BSPs are defined in clauses A.2, A.3, and A.4, respectively.

NOTE: Conformance of BFPs is not addressed in this part of ISO/IEC 19784, but is specified in subsequent parts of this multi-part standard. Interfaces to BioAPI Units are not standardized.

A.2 BioAPI Conformant Biometric Application

A.2.1 To claim compliance to the BioAPI specification, a biometric application shall, for each BioAPI function call utilized, invoke that operation consistently with this specification. That is, all input parameters shall be present and valid. The application shall accept all valid output parameters and return values.

A.2.2 There is no minimum set of functions that are required to be called. However, the biometric application shall conform to the call dependencies identified for the functions.

A.3 BioAPI Conformant Framework

A.3.1 The BioAPI Framework component is the middleware infrastructure between BioAPI conformant applications and BSPs. It serves the following general purposes:

a) BSP loading/attaching

b) BSP and BFP management

c) Component registry maintenance and management

d) Function call pass-through/API-SPI translation

e) Handling of event notifications from BSPs, and sending those event notifications to (possibly multiple) event handlers in applications that have loaded that BSP.

f) Supporting API calls related to the installation or de-installation of BioAPI components, with appropriate update of the component registry.

g) Supporting queries from a BSP about installed BFPs

A.3.2 To claim conformance to the BioAPI specification, a BioAPI Framework shall:

a) Provide component management functions as specified in clause 8.1.

b) Provide component registry services in accordance with clause 10.

c) Perform translation of API functions specified in clause 8 into the corresponding SPI functions specified in clause 9.

d) Conform to the data structures as defined in clause 7 and the error codes as defined in clause 11 when implementing a) through c), above.

e) Handle event notifications and callbacks as defined in clauses 7.28, 8.3, and 9.2.

A.3.3 A conformant BioAPI Framework is required to support ALL options identified in this standard, since it will provide services to applications and BSPs that may implement any of those options. No subsets of conformant Framework functions are defined.

A.4 BioAPI Conformant BSPs

To claim conformance to the BioAPI specification, BSPs shall implement mandatory functions for their conformance sub-class, as defined below, in accordance with the SPI defined in clause 9. BSPs claim conformance to one of the conformance sub-classes specified in clause A.1.1.

BSPs shall accept all valid input parameters and return valid outputs. Optional capabilities and returns are not required to claim conformance; but any optional functions or parameters that are implemented shall be implemented in accordance with the specification requirements. Additional parameters shall not be required.

The BSP installation process shall perform the population of all required component registry entries.

BSPs shall possess a valid and unique UUID that is associated with a specific BSP product and version.

NOTE: The UUID may be self-generated (see ISO 9834-8) and should (but need not) be the same on multiple systems where the same BSP product/version is installed.

BIRs generated by the BSP shall conform to the data structures defined in clause 7 (they shall be BioAPI BIRs). BSPs shall only return BioAPI_BIR data containing a registered FormatOwner with an associated valid FormatType (see clause 7.6 and Annex B).

BSPs shall perform error handling as defined in clause 11.

All BSPs shall support basic Component Management (clause 9.3.1), Handle (clause 9.3.2), Event (9.3.3.1) and Utility (clause 9.3.7) operations. Callback (clause 9.3.3.2), Database (clause 9.3.5), and BioAPI Unit (clause 9.3.6) operations are optional.

The following table is a summary of BSP conformance requirements by subclass of BSP. Details are provided in the following sub-clauses. Clause A.4.6 addressed conformance with respect to optional capabilities.

Table A.1 — BSP Conformance Sub-Classes

Function	Verification BSP	Identification BSP	Capture BSP	Verification Engine	Identification Engine
Component Management Functions					
BioSPI_BSPLoad	X	X	X	X	X
BioSPI_BSPUnload	X	X	X	X	X
BioSPI_BSPAttach	X	X	X	X	X
BioSPI_BSPDetach	X	X	X	X	X
BioSPI_QueryUnits	X	X	X		
BioSPI_QueryBFPs					
BioSPI_ControlUnit					
Handle Functions					
BioSPI_FreeBIRHandle	X	X	X	X	X
BioSPI_GetBIRFromHandle	X	X	X	X	X
BioSPI_GetHeaderFromHandle	X	X	X	X	X
Callback and Event Functions					
BioSPI_EnableEvents	X	X	X	X	X
BioSPI_SetGUICallbacks					
Biometric Functions					
BioSPI_Capture			X		
BioSPI_CreateTemplate				X	X
BioSPI_Process				X	X
BioSPI_ProcessWithAuxBIR					
BioSPI_VerifyMatch				X	X
BioSPI_IdentifyMatch					X
BioSPI_Enroll	X	X			
BioSPI_Verify	X	X			
BioSPI_Identify		X			
BioSPI_Import					
BioSPI_PresetIdentifyPopulation					
Database Functions					
BioSPI_DbOpen					
BioSPI_DbClose					
BioSPI_DbCreate					
BioSPI_DbDelete					
BioSPI_DbSetMarker					

Function	Verification BSP	Identification BSP	Capture BSP	Verification Engine	Identification Engine
BioSPI_DbFreeMarker					
BioSPI_DbStoreBIR					
BioSPI_DbGetBIR					
BioSPI_DbGetNextBIR					
BioSPI_DbDeleteBIR					
BioAPI Unit Functions					
BioSPI_SetPowerMode					
BioSPI_SetIndicatorStatus					
BioSPI_GetIndicatorStatus					
BioSPI_CalibrateSensor					
Utility Functions					
BioSPI_Cancel	X	X	X	X	X
BioSPI_Free	X	X	X	X	X

A.4.1 BioAPI Conformant Verification BSPs

Verification BSPs are those which are capable of performing 1:1 matching (or authentication), but not 1:N identification matching.

NOTE: 1:few (one-to-few) matching may be supported as a series of 1:1 calls.

A BioAPI conformant Verification BSP shall support the following biometric functions:

- ***BioSPI_Verify***
- ***BioSPI_Enroll***

A.4.1.1 ***BioSPI_Verify*****.** Only the nearest, better supported *MaxFMRRequested* is required to be supported; however, the BSP shall return that supported value (*FMRAchieved*). (The BSP shall indicate in the component registry whether it implements coarse scoring.)

NOTE: See clause C.4 for an explanation of use of FMR for scoring and thresholding.

Acceptance and use of *Subtype* is optional.

Return of *Payload* is required only if one is associated with the input *ReferenceTemplate* and if the score sufficiently exceeds the *FMRAchieved* (the BSP shall post minimum FMR required to return payload in the component registry).

Return of *AdaptedBIR* is optional. Return of raw data (*AuditData*) is optional.

All BSPs shall provide any necessary user interface (as a default user interface) associated with the capture portion of the ***Verify*** operation. Support for application control of a GUI is optional.

A.4.1.2 ***BioSPI_Enroll.*** The *Purpose* value BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY shall be accepted. If a different purpose value is requested but not supported, an error condition shall be set as a BioAPI_RETURN.

Acceptance of *Payload* is optional (i.e., if a payload is provided, it doesn't have to be accepted if the BSP does not support payload carry).

Acceptance and use of *Subtype* is optional.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input *OutputFormat* and return the *NewTemplate* in that format.

Use of the input *ReferenceTemplate*, when provided, to create the output *NewTemplate* is optional.

Return of raw data (*AuditData*) is optional.

All BSPs shall provide any necessary user interface (as a default user interface) associated with the capture portion of the ***Enroll*** operation. Support for application control of a GUI is optional.

A.4.2 BioAPI Conformant Identification BSPs

Identification BSPs are those which are capable of performing both 1:N identification matching as well as 1:1 matching (or verification). A BioAPI conformant Identification BSP shall support the following biometric functions:

- ***BioSPI_Verify***
- ***BioSPI_Identify***
- ***BioSPI_Enroll***

A.4.2.1 ***BioSPI_Verify***. Only the nearest, better supported *MaxFMRRequested* must be supported; however, the BSP shall return that supported value (*FMRAchieved*). (The BSP shall indicate in component registry whether it implements coarse scoring.)

NOTE: See clause C.4 for an explanation of the use of FMR for scoring and thresholding.

Acceptance and use of *Subtype* is optional.

Return of *Payload* is required only if one is associated with the input *ReferenceTemplate* and if the score sufficiently exceeds the *FMRAchieved* (the BSP shall post minimum FMR required to return payload in the component registry).

Return of *AdaptedBIR* is optional. Return of raw data (*AuditData*) is optional.

As a default, all BSPs shall provide any GUI associated with the capture portion of the ***Verify*** operation. However, support for application control of the GUI is optional.

A.4.2.2 ***BioSPI_Identify.*** Only the nearest, better supported *MaxFMRRequested* must be supported. Return of matching *Candidates* is required; however, the BSP may return values for the *FMRAchieved* field as next nearest step/increment. (The BSP shall indicate in component registry whether it implements coarse scoring.)

NOTE: See clause C.4 for an explanation of the use of FMR for scoring and thresholding.

Acceptance and use of *Subtype* is optional.

Support of binning is optional.

Return of raw data (*AuditData*) is optional.