
**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)

AVA Documents
Document Sharing Hub
© ISO/IEC 2006

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 2006

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

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.....	vi
Introduction	vii
1 Scope	1
2 Conformance	2
3 Normative references	2
4 Terms and definitions.....	2
5 Symbols and abbreviated terms	7
6 The BioAPI architecture	8
6.1 The BioAPI API/SPI Architectural Model	8
6.2 The BioAPI BSP Architectural Model.....	9
6.3 The Component Registry	10
6.4 BSP and BFP Installation and De-installation	11
6.5 BSP Load and BioAPI Unit Attachment.....	12
6.6 Controlling BioAPI Units.....	13
6.7 BIR Structure and Handling.....	13
6.7.1 BIR Structure.....	13
6.7.2 BIR Data Handling.....	14
7 BioAPI types and macros	15
7.1 BioAPI	15
7.2 BioAPI_BFP_LIST_ELEMENT	15
7.3 BioAPI_BFP_SCHEMA	15
7.4 BioAPI_BIR	16
7.5 BioAPI_BIR_ARRAY_POPULATION	17
7.6 BioAPI_BIR_BIOMETRIC_DATA_FORMAT	17
7.7 BioAPI_BIR_BIOMETRIC_PRODUCT_ID	17
7.8 BioAPI_BIR_BIOMETRIC_TYPE	18
7.9 BioAPI_BIR_DATA_TYPE	18
7.10 BioAPI_BIR_HANDLE	19
7.11 BioAPI_BIR_HEADER.....	19
7.12 BioAPI_BIR_PURPOSE	20
7.13 BioAPI_BIR_SECURITY_BLOCK_FORMAT	21
7.14 BioAPI_BIR_SUBTYPE	21
7.15 BioAPI_BOOL.....	22
7.16 BioAPI_BSP_SCHEMA	22
7.17 BioAPI_CANDIDATE	23
7.18 BioAPI_CATEGORY	24
7.19 BioAPI_DATA	24
7.20 BioAPI_DATE	24
7.21 BioAPI_DB_ACCESS_TYPE	25
7.22 BioAPI_DB_MARKER_HANDLE	25
7.23 BioAPI_DB_HANDLE	25
7.24 BioAPI_DBBIR_ID	25
7.25 BioAPI_DTG.....	25
7.26 BioAPI_EVENT	26
7.27 BioAPI_EVENT_MASK.....	26
7.28 BioAPI_EventHandler	26
7.29 BioAPI_FMR	27
7.30 BioAPI_FRAMEWORK_SCHEMA	27
7.31 BioAPI_GUI_BITMAP	28

7.32	BioAPI_GUI_MESSAGE	28
7.33	BioAPI_GUI_PROGRESS	29
7.34	BioAPI_GUI_RESPONSE	29
7.35	BioAPI_GUI_STATE	29
7.36	BioAPI_GUI_STATE_CALLBACK	29
7.37	BioAPI_GUI_STREAMING_CALLBACK	30
7.38	BioAPI_HANDLE	30
7.39	BioAPI_IDENTIFY_POPULATION	31
7.40	BioAPI_IDENTIFY_POPULATION_TYPE	31
7.41	BioAPI_INDICATOR_STATUS	31
7.42	BioAPI_INPUT_BIR	31
7.43	BioAPI_INPUT_BIR_FORM	32
7.44	BioAPI_INSTALL_ACTION	32
7.45	BioAPI_INSTALL_ERROR	32
7.46	BioAPI_OPERATIONS_MASK	32
7.47	BioAPI_OPTIONS_MASK	33
7.48	BioAPI_POWER_MODE	34
7.49	BioAPI_QUALITY	34
7.50	BioAPI_RETURN	35
7.51	BioAPI_STRING	35
7.52	BioAPI_TIME	36
7.53	BioAPI_UNIT_ID	36
7.54	BioAPI_UNIT_LIST_ELEMENT	36
7.55	BioAPI_UNIT_SCHEMA	36
7.56	BioAPI_UUID	38
7.57	BioAPI_VERSION	38
8	BioAPI functions	39
8.1	Component Management Functions	39
8.1.1	BioAPI_Init	39
8.1.2	BioAPI_Terminate	40
8.1.3	BioAPI_GetFrameworkInfo	41
8.1.4	BioAPI_EnumBSPs	42
8.1.5	BioAPI_BSPLoad	43
8.1.6	BioAPI_BSPUnload	45
8.1.7	BioAPI_BSPAttach	46
8.1.8	BioAPI_BSPDetach	48
8.1.9	BioAPI_QueryUnits	49
8.1.10	BioAPI_EnumBFPs	51
8.1.11	BioAPI_QueryBFPs	52
8.1.12	BioAPI_ControlUnit	54
8.2	Data Handle Operations	55
8.2.1	BioAPI_FreeBIRHandle	55
8.2.2	BioAPI_GetBIRFromHandle	56
8.2.3	BioAPI_GetHeaderFromHandle	57
8.3	Callback and Event Operations	58
8.3.1	BioAPI_EnableEvents	58
8.3.2	BioAPI_SetGUICallbacks	59
8.4	Biometric Operations	60
8.4.1	BioAPI_Capture	60
8.4.2	BioAPI_CreateTemplate	62
8.4.3	BioAPI_Process	64
8.4.4	BioAPI_ProcessWithAuxBIR	65
8.4.5	BioAPI_VerifyMatch	67
8.4.6	BioAPI_IdentifyMatch	69
8.4.7	BioAPI_Enroll	72
8.4.8	BioAPI_Verify	74
8.4.9	BioAPI_Identify	76
8.4.10	BioAPI_Import	79
8.4.11	BioAPI_PresetIdentifyPopulation	80

8.5	Database Operations	81
8.5.1	BioAPI_DbOpen	81
8.5.2	BioAPI_DbClose	83
8.5.3	BioAPI_DbCreate	84
8.5.4	BioAPI_DbDelete	85
8.5.5	BioAPI_DbSetMarker	86
8.5.6	BioAPI_DbFreeMarker	87
8.5.7	BioAPI_DbStoreBIR	88
8.5.8	BioAPI_DbGetBIR	89
8.5.9	BioAPI_DbGetNextBIR	90
8.5.10	BioAPI_DbDeleteBIR	91
8.6	BioAPI Unit operations	92
8.6.1	BioAPI_SetPowerMode	92
8.6.2	BioAPI_SetIndicatorStatus	93
8.6.3	BioAPI_GetIndicatorStatus	94
8.6.4	BioAPI_CalibrateSensor	95
8.7	Utility Functions	96
8.7.1	BioAPI_Cancel	96
8.7.2	BioAPI_Free	97
9	BioAPI Service Provider Interface	98
9.1	Summary	98
9.2	Type Definitions for Biometric Service Providers	98
9.2.1	BioSPI_EventHandler	98
9.2.2	BioSPI_BFP_ENUMERATION_HANDLER	99
9.2.3	BioSPI_MEMORY_FREE_HANDLER	100
9.3	Biometric Service Provider Operations	101
9.3.1	SPI Component Management Operations	101
9.3.2	SPI Data Handle Operations	107
9.3.3	SPI Callback and Event Operations	108
9.3.4	SPI Biometric Operations	109
9.3.5	SPI Database Operations	112
9.3.6	SPI BioAPI Unit operations	114
9.3.7	SPI Utility Functions	115
10	Component registry interface	116
10.1	BioAPI Registry Schema	116
10.1.1	Framework Schema	116
10.1.2	BSP Schema	117
10.1.3	BFP Schema	118
10.2	Component registry functions	119
10.2.1	BioAPI_Util_InstallBSP	119
10.2.2	BioAPI_Util_InstallBFP	120
11	BioAPI error handling	121
11.1	Error Values and Error Codes Scheme	121
11.2	Error Codes and Error Value Enumeration	121
11.2.1	BioAPI Error Value Constants	121
11.2.2	Implementation-Specific Error Codes	121
11.2.3	General Error Codes	121
11.2.4	Component Management Error Codes	123
11.2.5	Database Error Values	124
11.2.6	Location Error Values	125
11.2.7	Quality Error Codes	127
Annex A (normative)	Conformance	128
Annex B (normative)	CBEFF Patron Format Specification: BioAPI patron format	140
Annex C (informative)	Specification overview	145
Annex D (informative)	Calling sequence examples and sample code	154
Bibliography		167

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 from 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.

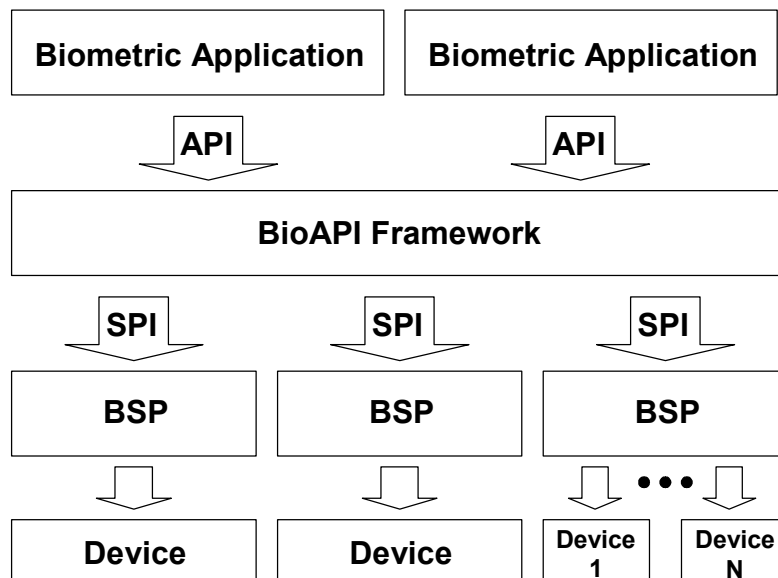


Figure 1 — BioAPI API/SPI Model

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.

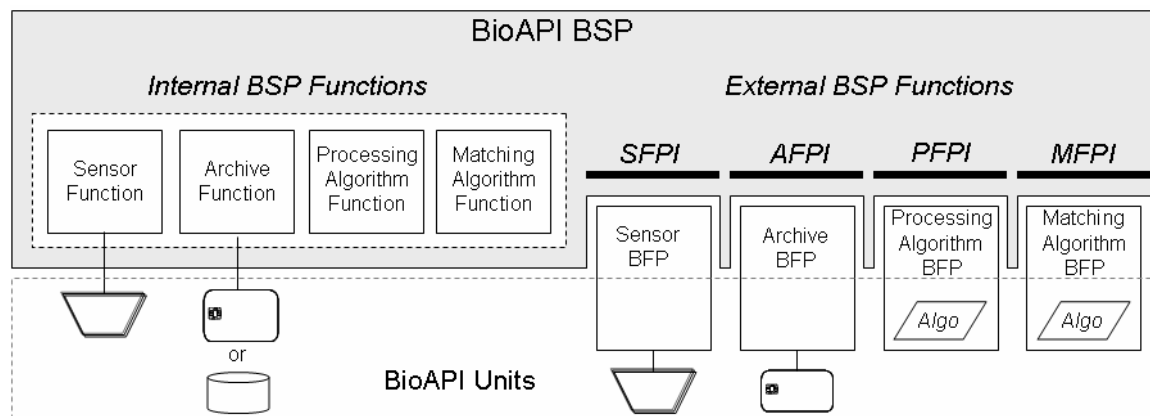


Figure 2 — Illustration of BSP architecture

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) of 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.

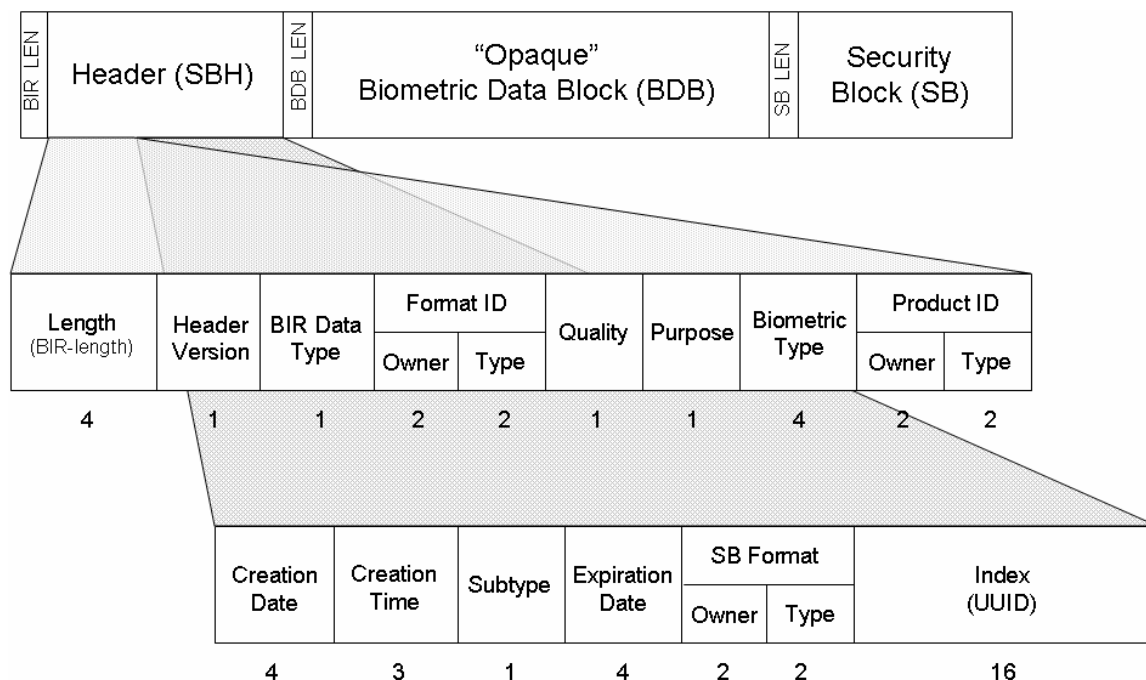


Figure 3 — Biometric information record (BIR)

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:

- by reference to its handle (if it is in the BSP being invoked);
- by reference to its key value (UUID) in an open BIR database (controlled by the BSP being invoked); or
- 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 specified 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_KEYSTROKE_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)
#define BioAPI_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_IDENTIFICATION_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 management 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_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).