INTERNATIONAL STANDARD

ISO/IEC 19784-1

Second edition
2018-04

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 19784-1:2018(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword ... **viii**
Introduction ... **ix**
1 Scope ... **1**
2 Normative references ... **2**
3 Terms and definitions ... **2**
4 Symbols and abbreviated terms ... **8**
5 Conformance ... **8**
6 The BioAPI architecture ... **9**
6.1 The full BioAPI API/SPI Architectural Model ... 9
6.2 The framework-free BioAPI Architectural Model ... 10
6.3 The BioAPI BSP Architectural Model ... 10
6.4 The component registry ... 12
6.5 BSP and BFP Installation and De-installation ... 13
6.6 BSP Load and BioAPI Unit Attachment ... 14
6.7 Controlling BioAPI Units ... 15
6.8 BIR Structure and Handling ... 15
6.8.1 BIR Structure ... 15
6.8.2 BIR Data Handling ... 16
7 BioAPI types and macros ... **17**
7.1 BioAPI ... 17
7.2 BioAPI_ACBio_PARAMETERS (BioAPI 2.2) ... 17
7.3 BioAPI_ASN1_BIR (BioAPI 2.2) ... 17
7.4 BioAPI_ASN1_ENCODED (BioAPI 2.2) ... 18
7.5 BioAPI_BFP_LIST_ELEMENT ... 18
7.6 BioAPI_BFP_SCHEMA ... 18
7.7 BioAPI_BIR ... 19
7.8 BioAPI_BIR_ARRAY_POPULATION ... 19
7.9 BioAPI_BIR_BIOMETRIC_DATA_FORMAT ... 20
7.10 BioAPI_BIR_BIOMETRIC_PRODUCT_ID ... 20
7.11 BioAPI_BIR_BIOMETRIC_TYPE (BioAPI 2.0) ... 20
7.12 BioAPI_BIR_BIOMETRIC_TYPE (BioAPI 2.1) ... 21
7.13 BioAPI_BIR_DATA_TYPE ... 22
7.14 BioAPI_BIR_HANDLE ... 23
7.15 BioAPI_BIR_HEADER ... 23
7.16 BioAPI_BIR_PURPOSE ... 24
7.17 BioAPI_BIR_SECURITY_BLOCK_FORMAT ... 25
7.18 BioAPI_BIR_SUBTYPE (BioAPI 2.0) ... 25
7.19 BioAPI_BIR_SUBTYPE (BioAPI 2.1) ... 26
7.20 BioAPI_BIR_SUBTYPE_MASK (BioAPI 2.1) ... 27
7.21 BioAPI_BOOL ... 28
7.22 BioAPI_BSP_SCHEMA (BioAPI 2.0) ... 28
7.23 BioAPI_BSP_SCHEMA (BioAPI 2.1) ... 30
7.24 BioAPI_CANDIDATE ... 32
7.25 BioAPI_CATEGORY ... 32
7.26 BioAPI_DATA ... 32
7.27 BioAPI_DATE ... 32
7.28 BioAPI_DB_ACCESS_TYPE ... 33
7.29 BioAPI_DB_MARKER_HANDLE ... 33
7.30 BioAPI_DB_HANDLE ... 33
7.31 BioAPI_DBBIR_ID ... 33
7.32 BioAPI_DTG ... 34
7.33 BioAPI_ENCRYPTION_ALG (BioAPI 2.2) ... 34

7.34 BioAPI_ENCRYPTION_INFO (BioAPI 2.2) 34
7.35 BioAPI_ERROR_INFO (BioAPI 2.1) 34
7.36 BioAPI_EVENT 35
7.37 BioAPI_EVENT_MASK 35
7.38 BioAPI_EventHandler 35
7.39 BioAPI_FMR 36
7.40 BioAPI_FRAMEWORK_SCHEMA 36
7.41 BioAPI_GUI_BITMAP (BioAPI 2.0) 37
7.42 BioAPI_GUI_BITMAP (BioAPI 2.1) 37
7.43 BioAPI_GUI_BITMAP_ARRAY (BioAPI 2.1) 38
7.44 BioAPI_GUI_ENROLL_TYPE (BioAPI 2.1) 38
7.45 BioAPI_GUI_EVENT_SUBSCRIPTION (BioAPI 2.1) 39
7.46 BioAPI_GUI_MESSAGE (BioAPI 2.0) 39
7.47 BioAPI_GUI_MOMENT (BioAPI 2.1) 40
7.48 BioAPI_GUI_OPERATION (BioAPI 2.1) 41
7.49 BioAPI_GUI_PROGRESS (BioAPI 2.0) 41
7.50 BioAPI_GUI_PROGRESS (BioAPI 2.1) 42
7.51 BioAPI_GUI_RESPONSE (BioAPI 2.0) 42
7.52 BioAPI_GUI_RESPONSE (BioAPI 2.1) 42
7.53 BioAPI_GUI_STATE (BioAPI 2.0) 44
7.54 BioAPI_GUI_STATE_CALLBACK (BioAPI 2.0) 44
7.55 BioAPI_GUI_STREAMING_CALLBACK (BioAPI 2.0) 45
7.56 BioAPI_GUI_SUBOPERATION (BioAPI 2.1) 46
7.57 BioAPI_HANDLE 47
7.58 BioAPI_HASH_ALG (BioAPI 2.2) 47
7.59 BioAPI_IDENTIFY_POPULATION 47
7.60 BioAPI_IDENTIFY_POPULATION_TYPE 47
7.61 BioAPI_INDICATOR_STATUS 48
7.62 BioAPI_INPUT_BIR 48
7.63 BioAPI_INPUT_BIR_FORM 48
7.64 BioAPI_INSTALL_ACTION 48
7.65 BioAPI_INSTALL_ERROR 48
7.66 BioAPI_KEY_INFO (BioAPI 2.2) 49
7.67 BioAPI_KEY_TRANSPORT (BioAPI 2.2) 49
7.68 BioAPI_MAC_ALG (BioAPI 2.2) 49
7.69 BioAPI_MAC_INFO (BioAPI 2.2) 49
7.70 BioAPI_OPERATIONS_MASK 50
7.71 BioAPI_OPTIONS_MASK 50
7.72 BioAPI_POWER_MODE 52
7.73 BioAPI_QUALITY 52
7.74 BioAPI_RETURN 53
7.75 BioAPI_SECURITY_OPTIONS_MASK (BioAPI 2.2) 53
7.76 BioAPI_SECURITY_PROFILE (BioAPI 2.2) 53
7.77 BioAPI_DIGITAL_SIGNATURE_ALG (BioAPI 2.2) 54
7.78 BioAPI_STRING 55
7.79 BioAPI_TIME 55
7.80 BioAPI_UNIT_ID 55
7.81 BioAPI_UNIT_LIST_ELEMENT 55
7.82 BioAPI_UNIT_SCHEMA 55
7.83 BioAPI_UNIT_SCHEMA (BioAPI 2.2) 57
7.84 BioAPI_UUID 58
7.85 BioAPI_VERSION 58
7.86 GUI Events 58
7.86.1 BioAPI_GUI_SELECT_EVENT_HANDLER (BioAPI 2.1) 59
7.86.2 BioAPI_GUI_STATE_EVENT_HANDLER (BioAPI 2.1) 61
7.86.3 BioAPI_GUI_PROGRESS_EVENT_HANDLER (BioAPI 2.1) 63

8 BioAPI functions 65
8.1 Component Management Functions 65

8.1.1 BioAPI_Init 65
8.1.2 BioAPI_Terminate 66
8.1.3 BioAPI_GetFrameworkInfo 66
8.1.4 BioAPI_EnumBSPs 67
8.1.5 BioAPI_BSPLoad 67
8.1.6 BioAPI_BSPUnload 69
8.1.7 BioAPI_BSPAttach 70
8.1.8 BioAPI_BSPAttachSecure (BioAPI 2.2) 71
8.1.9 BioAPI_BSPDetach 73
8.1.10 BioAPI_QueryUnits 74
8.1.11 BioAPI_EnumBFPs 75
8.1.12 BioAPI_QueryBFPs 76
8.1.13 BioAPI_ControlUnit 77
8.1.14 BioAPI_Control (BioAPI 2.1) 77
8.1.15 BioAPI_Transform (BioAPI 2.1) 78
8.1.16 BioAPI_LinkToEndpoint (BioAPI 2.1) 80
8.1.17 BioAPI_UnlinkFromEndpoint (BioAPI 2.1) 80
8.1.18 BioAPI_EnumFrameworks (BioAPI 2.1) 81
8.2 Data Handle Operations 82
8.2.1 BioAPI_FreeBIRHandle 82
8.2.2 BioAPI_GetBIRFromHandle 82
8.2.3 BioAPI_GetHeaderFromHandle 83
8.3 Callback and Event Operations 83
8.3.1 BioAPI_EnableEvents 83
8.3.2 BioAPI_SetGUICallbacks (BioAPI 2.0) 84
8.3.3 BioAPI_NotifyGUIProgressEvent (BioAPI 2.1) 85
8.3.4 BioAPI_NotifyGUISelectEvent (BioAPI 2.1) 86
8.3.5 BioAPI_NotifyGUIStateEvent (BioAPI 2.1) 87
8.3.6 BioAPI_QueryGUIEventSubscriptions (BioAPI 2.1) 88
8.3.7 BioAPI_RedirectGUIEvents (BioAPI 2.1) 89
8.3.8 BioAPI_SubscribeToGUIEvents (BioAPI 2.1) 91
8.3.9 BioAPI_UnredirectGUIEvents (BioAPI 2.1) 93
8.3.10 BioAPI_UnsubscribeFromGUIEvents (BioAPI 2.1) 94
8.3.11 BioAPI_EnableEventNotifications (BioAPI 2.1) 95
8.4 Biometric Operations 95
8.4.1 BioAPI_Capture 95
8.4.2 BioAPI_CreateTemplate 97
8.4.3 BioAPI_Process 99
8.4.4 BioAPI_ProcessWithAuxBIR (BioAPI2.0 and BioAPI2.1) 100
8.4.5 BioAPI_ProcessUsingAuxBIRs (BioAPI 2.2) 101
8.4.6 BioAPI_VerifyMatch 102
8.4.7 BioAPI_VerifyMatchUsingAuxBIRs (BioAPI 2.2) 104
8.4.8 BioAPI_IdentifyMatch 106
8.4.9 BioAPI_Decide (BioAPI 2.2) 109
8.4.10 BioAPI_Fuse (BioAPI 2.2) 110
8.4.11 BioAPI_Enroll 111
8.4.12 BioAPI_Verify 113
8.4.13 BioAPI_Identify 115
8.4.14 BioAPI_Import 118
8.4.15 BioAPI_Export (BioAPI 2.2) 119
8.4.16 BioAPI_PresetIdentifyPopulation 120
8.5 Database Operations 121
8.5.1 BioAPI_DbOpen 121
8.5.2 BioAPI_DbClose 122
8.5.3 BioAPI_DbCreate 122
8.5.4 BioAPI_DbDelete 123
8.5.5 BioAPI_DbSetMarker 124
8.5.6 BioAPI_DbFreeMarker 124

8.5.7 BioAPI_DbStoreBIR ... 125
8.5.8 BioAPI_DbGetBIR ... 126
8.5.9 BioAPI_DbGetNextBIR ... 127
8.5.10 BioAPI_DbDeleteBIR ... 128
8.6 BioAPI Unit operations ... 128
8.6.1 BioAPI_SetPowerMode ... 128
8.6.2 BioAPI_SetIndicatorStatus ... 129
8.6.3 BioAPI_GetIndicatorStatus ... 130
8.6.4 BioAPI_CalibrateSensor ... 130
8.7 Utility Functions ... 131
8.7.1 BioAPI_Cancel ... 131
8.7.2 BioAPI_Free ... 131

9 BioAPI Service Provider Interface ... 132
9.1 Summary ... 132
9.2 Type Definitions for Biometric Service Providers ... 132
9.2.1 BioSPI_EventHandler ... 132
9.2.2 BioSPI_BFP_ENUMERATION_HANDLER ... 133
9.2.3 BioSPI_MEMORY_FREE_HANDLER ... 134
9.2.4 BioSPI_GUI_PROGRESS_EVENT_HANDLER (BioAPI 2.1) ... 135
9.2.5 BioSPI_GUI_SELECT_EVENT_HANDLER (BioAPI 2.1) ... 136
9.2.6 BioSPI_GUI_STATE_EVENT_HANDLER (BioAPI 2.1) ... 137
9.3 Biometric Service Provider Operations ... 137
9.3.1 SPI Component Management Operations ... 137
9.3.2 SPI Data Handle Operations ... 143
9.3.3 SPI Callback and Event Operations ... 143
9.3.4 SPI Biometric Operations ... 145
9.3.5 SPI Database Operations ... 148
9.3.6 SPI BioAPI Unit operations ... 150
9.3.7 SPI Utility Functions ... 151

10 Component registry interface ... 151
10.1 BioAPI Registry Schema ... 152
10.1.1 Framework Schema ... 152
10.1.2 BSP Schema ... 152
10.1.3 BFP Schema ... 154
10.2 Component registry functions ... 155
10.2.1 BioAPI_Util_InstallBSP ... 155
10.2.2 BioAPI_Util_InstallBFP ... 156
10.2.3 BioAPI_RegisterBSP (BioAPI 2.1) ... 156
10.2.4 BioAPI_UnregisterBSP (BioAPI 2.1) ... 157
10.2.5 BioAPI_RegisterBFP (BioAPI 2.1) ... 158
10.2.6 BioAPI_UnregisterBFP (BioAPI 2.1) ... 159
10.2.7 BioAPI_GetLastErrorInfo (BioAPI 2.1) ... 159

11 BioAPI error handling ... 160
11.1 Error Values and Error Codes Scheme ... 160
11.2 Error Codes and Error Value Enumeration ... 160
11.2.1 BioAPI Error Value Constants ... 160
11.2.2 Implementation-Specific Error Codes ... 160
11.2.3 General Error Codes ... 161
11.2.4 Component Management Error Codes ... 162
11.2.5 Database Error Values ... 163
11.2.6 Location Error Values ... 163
11.2.7 Quality Error Codes ... 165
11.2.8 Security Error Codes (BioAPI 2.2) ... 166

Annex A (normative) **Conformance ... 168**

Annex B (normative) **CBEFF Patron Format Specification: BioAPI patron format ... 185**

Annex C (informative) **Specification overview** **191**
Annex D (informative) **Calling sequence examples and sample code** **214**
Annex E (normative) **ASN.1 specification of BioAPI_BIR (BioAPI2.2)** **232**
Bibliography **234**

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

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. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This document was prepared by ISO/IEC JTC 1, *Information technology*, SC 37, *Biometrics*.

This second edition cancels and replaces the first edition (ISO/IEC 19784-1:2006). It also incorporates the Amendments ISO/IEC 19784-1:2006/Amd 1:2007, ISO/IEC 19784-1:2006/Amd 2:2009 and ISO/IEC 19784-1:2006/Amd 3:2010.

A list of all the parts in the ISO 19784 series, can be found on the ISO website.

Introduction

This document provides a high-level generic biometric authentication model suited to most forms of biometric technology. 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.

However, this document can also be applied where a system is to be built from conforming BSP components (without a BioAPI Framework module), using platform-specific system-integration mechanisms - see Clause 6.

This document specifies the behaviour of the BioAPI Framework when applications and BSPs are in the same system. Other interworking standards (see 4.29) specify modifications of that behaviour that enable both BSPs and Graphical User Interfaces to be remote from the system containing an application.

NOTE 1 ISO/IEC 24708 BioAPI Interworking Protocol (BIP)[6] is an example of an interworking standard.

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 2 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 document 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 documnet 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 shall be supported.

NOTE 3 A separate IS, ISO/IEC 24709, addresses conformance testing for this BioAPI specification[9].

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.

Annex E provides an ASN.1 specification of the BioAPI Biometric Information Record (BIR).

This revision is a merged document of ISO/IEC 19784-1: 2006, ISO/IEC 19784-1: 2006/Amd 1: 2007, ISO/IEC 19784-1: 2006/Amd 2:2009, and ISO/IEC 19784-1: 2006/Amd 3 2010. But in this document, the defects in ISO/IEC 19784-1: 2006/Amd 3 are corrected. BioAPI 2.0 means specification from ISO/IEC 19784-1: 2006/Amd 2:2009, BioAPI 2.1 from ISO/IEC 19784-1: 2006/Amd 1: 2007, and BioAPI 2.2 from ISO/IEC 19784-1: 2006/Amd 3 2010.

Information technology — Biometric application programming interface —

Part 1: BioAPI specification

1 Scope

This document 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 and to other International Standards.

For use in a system that does not include a BioAPI Framework (called a framework-free BioAPI system), only the SPI interface is applicable, with applications interfacing directly to that in a platform-specific manner.

NOTE 1 Many clauses and/or sub-clauses of this document are not applicable for implementation of a framework-free BioAPI system. These are identified at the head of the clause of sub-clause.

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 document 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 2 Where BioAPI Units are provided by a different vendor fom a BSP, a standardised BioAPI Function Provider Interface (FPI) may be needed. This is outside the scope of this document, but is specified by later parts for the different categories of FPI.

NOTE 3 Where a BioAPI Framework is not used in a system, the ability to support multiple applications and multiple BSPs is platform-dependent and depends on the nature of the system-integration techniques employed.

This document 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 document.

This document does not define security requirements for biometric applications and biometric service providers.

NOTE 4 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 document. Trade-offs between interoperability and performance are not in the scope of this document.

This document 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. It also specifies a version number described as Major 2, Minor 1, or version 2.1 that provides an enhanced Graphical User Interface. It also specifies a version

number described as Major 2, Minor 2, or version 2.2 that provides features supporting fusion and security. Some clauses and sub-clauses apply only to one of these versions, some to two or more. This is identified at the head of the relevant clauses and sub-clauses.

NOTE 5 Earlier versions of the BioAPI specification were not International Standards.

NOTE 6 The differences between the requirements of the 2.0 specification and the 2.1 specification for framework-free operation relate only to the biometric type values and encodings.

Conformance requirements are specified in Clause 5.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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 10646:2017, *Information technology — Universal Coded Character Set (UCS)*

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

ISO/IEC 19785-4:2010, *Information technology — Common Biometric Exchange Formats Framework — Part 4: Security block format specifications*

3 Terms and definitions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— IEC Electropedia: available at http://www.electropedia.org/

— ISO Online browsing platform: available at https://www.iso.org/obp

NOTE 1 Function names and data element names are not included here, but are defined within the body of this document.

NOTE 2 Many of these terms and definitions are not relevant for the use of this document for framework-free BioAPI systems.

4.1
ACBio instance
report generated by a biometric processing unit (BPU) compliant to ISO/IEC 24761 to show the validity of the result of one or more subprocesses executed in the BPU

[SOURCE: ISO/IEC 24761]

4.2
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 1 to entry: This procedure is used to minimize the effects of template aging.

4.3
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

4.4
authentication context for biometrics
ACBio
international standard that specifies the form and encoding of ACBio instances

[SOURCE: ISO/IEC 24761]

4.5
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 1 to entry: BioAPI components include BioAPI applications, the BioAPI Framework, BSPs, and BFPs.

4.6
BioAPI Function Provider
BFP
component that manages one or more BioAPI Units of a specific category

Note 1 to entry: Interfaces to BioAPI Function Providers are standardized in subsequent parts of the ISO/IEC 19784 series.

Note 2 to entry: BFPs are categorized according to the categories of BioAPI Units that they manage (see 6.3.4).

4.7
BioAPI Unit
abstraction of a hardware or software resource that is directly managed by a BSP or BFP

Note 1 to entry: Note1 to entry: BioAPI Units are categorized (see 6.3.2) and include sensor units, archive units, matching algorithm units and processing algorithm units.

4.8
biographic data (BioAPI 2.2)
non-biometric data that potentially affects a biometric operation

4.9
biographic BIR (BioAPI 2.2)
non-biometric BIR that potentially affects a biometric operation

4.10
biometric
pertaining to the field of biometrics

4.11
Biometric Data Block
BDB
block of data with a defined format that contains one or more biometric samples or biometric templates

Note 1 to entry: Note1 to entry: ISO/IEC 19784 does not support BDB formats that are not an integral multiple of eight bits.

Note 2 to entry: There is no requirement that a BDB format be self-delimiting.

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

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

4.12
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

Note 1 to entry: See ISO/IEC 19785-1.

Note 2 to entry: This document defines a BIR format (see 7.7) 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 document, 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.

4.12.1
reference BIR
BIR whose BDB(s) contains one or more biometric templates

4.12.2
sample BIR
BIR whose BDB(s) contains only biometric samples that are not templates

4.13
biometric processing unit
BPU
entity that executes one or more subprocesses that perform a biometric verification at a uniform level of security

Note 1 to entry: See ISO/IEC 24761.

Note 2 to entry: A sensor, a smart card, and a comparison device are examples of BPUs.

4.14
biometric sample
information obtained from a biometric sensor, either directly or after further processing

Note 1 to entry: See also raw biometric sample, intermediate biometric sample, and processed biometric sample.

4.14.1
biometric template
biometric sample or combination of biometric samples that is suitable for storage as a reference for future comparison

4.14.2
intermediate biometric sample
biometric sample obtained by processing a raw biometric sample, intended for further processing

4.14.3
processed biometric sample
biometric sample suitable for comparison

4.14.4
raw biometric sample
biometric sample obtained directly from a biometric sensor

Note 1 to entry: 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.

4.14.5
reference template
biometric template that has been stored

4.15
biometric sensor
biometric hardware used to capture raw biometric samples from a subject

Note 1 to entry: The term 'biometric device' is used interchangeably with this term.

4.16
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

4.17
biometrics
automated recognition of individuals based on their behavioural and biological characteristics

4.18
BPU IO Index
integer assigned to each biometric data stream between BPUs by the subject, such as software, which utilizes the function of the BPU so that the validator can reconstruct the data flow among BPUs

Note 1 to entry: See ISO/IEC 24761.

4.19
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 1 to entry: 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.

4.20
component registry
information maintained by the BioAPI Framework concerning the BioAPI components that are available on a biometric system

4.21
decision BIR (BioAPI 2.2)
BIR which contains decision in the BDB

4.22
encrypt
encryption
(reversible) transformation of data by a cryptographic algorithm to produce ciphertext, i.e. hide the information content (protect the confidentiality) of the data

Note 1 to entry: 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 to entry: Encryption may be used for either security or privacy reasons.

4.23
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 1 to entry: The reference template would normally be stored by a biometric application, a BSP supporting an archive BioAPI Unit, or both.

4.24
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 to entry: Within BioAPI, FMR is used as a means of specifying scores and thresholds (see C.4).

Note 2 to entry: 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).

4.25
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 1 to entry: 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.

4.26
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 1 to entry: This is often called an "identification match" or "identifymatch".

4.27
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 to entry: An accept or reject decision would then normally be based upon whether this score exceeds a given threshold.

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

Note 3 to entry: See also identify (4.22) and verify (4.33).

4.28
payload
data, provided at the time of enrolment and associated with a reference template, which can be released upon a successful biometric verification

Note 1 to entry: Examples of payloads include user names, accounts, passwords, cryptographic keys, or digital certificates (see C.5).

4.29
score
score data
scoring
value indicating the degree of similarity or correlation between a biometric sample and a biometric reference template

4.30
score BIR (BioAPI 2.2)
BIR which contains score in the BDB

4.31
secure BioAPI (BioAPI 2.2)
BioAPI API and SPI interfaces that include the security features defined for version 2.2 of BioAPI

4.32
security block
block of data with a defined format that contains security information (for example, related to encryption or integrity) related to a BIR

Note 1 to entry: See ISO/IEC 19785-1.

4.33
self-contained device
combination device which includes a biometric sensor and all or part of the BSP functionality

Note 1 to entry: 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.

4.34
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 1 to entry: Digital signatures may be used for purposes of authentication, data integrity, and non-repudiation.

4.35
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

4.36
universally unique identifier
UUID
128-bit value generated in accordance with ISO/IEC 9834-8 and providing unique values between systems and over time

4.37
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 1 to entry: This is often called a "verification match" or "verifymatch".

4.38
interworking standards
standards that modify the behaviour of the BioAPI Framework (and BioAPI conformance requirements) to support the use of communications links to enable an application to interact with a remote BSP using a standardised protocol

4.39
test-verify
test-verification
one-to-one process of comparing a single biometric sample against a candidate of a biometric reference template at enrollment to determine whether the candidate template matches the test sample

4.40
enroll type
value denoting a pattern of suboperations performed by a BSP during an enroll operation

4 Symbols and abbreviated terms

ACBio	Authentication Context for Biometrics
API	Application Programming Interface
BDB	Biometric Data Block
BFP	BioAPI Function Provider
BIR	Biometric Information Record
BPU	Biometric Processing Unit
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
IRI	Internationalized Resource Identifier (see RFC 3987)
MAC	Message Authentication Code
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

5 Conformance

5.1 Annex A specifies the conformance requirements for BioAPI components claiming conformance to this document.

5.2 This document uses the C programming language (see ISO/IEC 9899) to specify the interfaces that it defines. A BioAPI component can conform to this document 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 document. (See also 7.1.)

6 The BioAPI architecture

6.1 The full BioAPI API/SPI Architectural Model

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

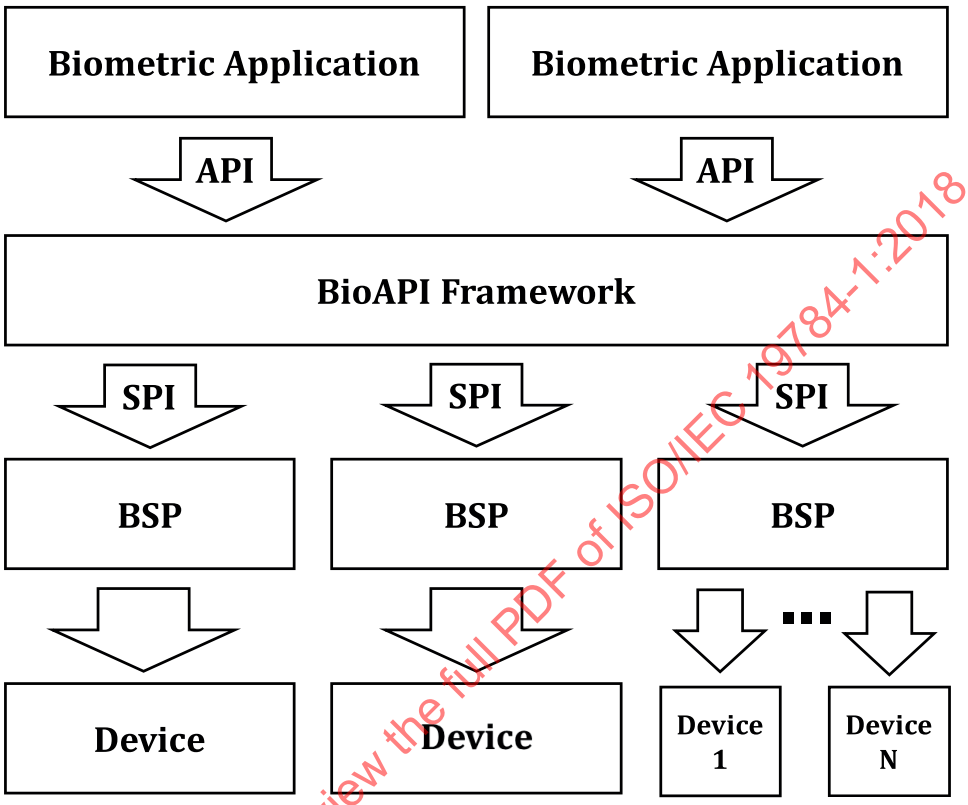

Figure 1 — BioAPI API/SPI Model

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 full 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 6.3.

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 8.1.7 (BioAPI 2.1 or less), and 8.1.8 (BioAPI 2.2 or greater). When the application no longer requires the use of the BSP, it is detached and unloaded (see clauses 8.1.6 and 8.1.9).

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 document support the presence in a biometric system of:

a) a singleBioAPI Framework with an associated component registry;

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

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

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 6.6) managed by that BSP;

e) providing graphical information to the application about an ongoing enrollment, verification, or identification operation.

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 framework-free BioAPI Architectural Model

6.2.1 The framework-free architecture recognises the use of BioAPI-conforming BSPs that provide (a defined subset of) the SPI interface for integration into systems in which other components are not standardised.

6.2.2 The conformance requirements in this case are reduced to the minimum necessary for reliable BSP operation, and apply only to BSPs.

6.2.3 The architecture within a BSP (see 6.3) still applies, but many BSPs intended for use in framework-free systems are likely to be (but need not be) monolithic implementations from a single vendor.

6.3 The BioAPI BSP Architectural Model

6.3.1 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.).

6.3.2 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;

— 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.

6.3.3 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.

6.3.4 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;

— processing algorithm BFP.

6.3.5 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.

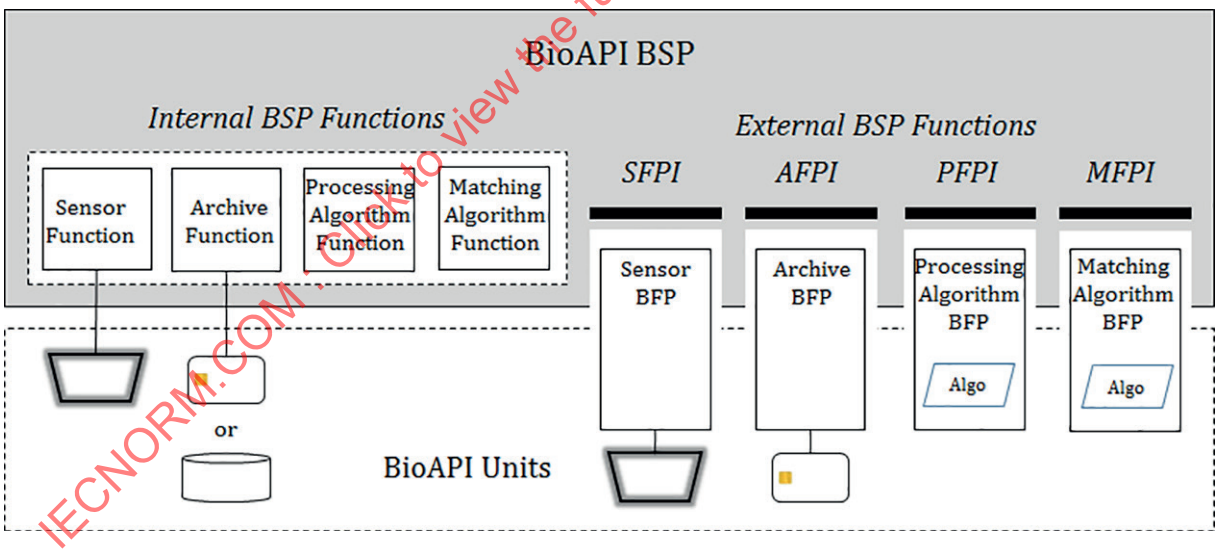

Figure 2 — Illustration of BSP architecture

6.3.6 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.

6.3.7 If the implementer of a BSP claims (see clause 7.71) 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.

6.3.8 There is a BioAPI API function and corresponding SPI function (and in some cases FPI functions) (see clauses 8.1.14 and 9.3.1.9) 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 6.7 for further details).

6.3.9 When a BSP attach session or a BSP attach secure session is established, at most one BioAPI Unit of each category is selected (see clause 8.1.7 and 8.1.8).

NOTE 1 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.

NOTE 2 At the time of standardizing this document, the FPIs for application of ISO/IEC 24761 are not specified yet. Therefore ISO/IEC 24761 is applicable only to the BSPs which implement internal BSP functions and do not have external BFPs attached because an ACBio instance shall be generated in a BioAPI Unit.

6.4 The component registry

NOTE The component registry function in a framework-free system is either not available, or is provided by non-standardised interfaces to platform-dependent functions. This sub-clause is not relevant for such systems.

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

6.4.2 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.

6.4.3 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.

6.4.4 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 8.1.11):

a) information about the BioAPI Framework itself;

b) details of all installed BSPs;

c) details of all installed BFPs.

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

6.4.5 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 8.1.12 and 8.1.10):

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

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).

6.4.6 The information listed in clauses 6.4.4 and 6.4.5 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*** or ***BioAPI_InitEndpoint*** (in BioAPI 2.1 or greater) (see clause 8.1.3);

b) details of all installed BSPs can be obtained at any time after ***BioAPI_Init*** or ***BioAPI_InitEndpoint*** (in BioAPI 2.1 ;r greater) (see clause 8.1.4);

c) details of all installed BFPs can be obtained at any time after ***Bi;API_Init*** or ***BioAPI_InitEndpoint*** (in BioAPI 2.1 or greater) (see clause 8.1.11);

d) details of all the installed BFPs that are supported by each BSP can be obtained at any time after ***BioAPI_BSPLoad*** (see clause 8.1.12) 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 8.1.10) of that BSP.

6.5 BSP and BFP Installation and De-installation

NOTE The installation and de-installation functions in a framework-free system are either not available, or are provided by non-standardised interfaces to platform-dependent functions. This sub-clause is not relevant for such systems.

6.5.1 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.

6.5.2 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 7.22 (BioAPI 2.0) and 7.23 (BioAPI 2.1 or greater).

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

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

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

6.5.4 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 8.1.11).

6.5.5 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 6.5.6) 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).

6.5.6 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 can also be under the control of other installed BFPs. This area is outside the standardization of BioAPI, and is again implementation-dependent.

6.6 BSP Load and BioAPI Unit Attachment

NOTE These functions in a framework-free system are either not available, or are provided by non-standardised interfaces to platform-dependent functions. This sub-clause is not relevant for such systems.

6.6.1 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 7.35) 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 1 '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 2 It is possible for an application to establish multiple simultaneous attach sessions with different BSPs (or even with the same BSP).

NOTE 3 Annex D provides sample calling sequence code.

6.6.2 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).

6.6.3 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.

6.6.4 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.

6.6.5 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.

6.6.6 On ***BioAPI_BSPAttach/BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater), 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.

6.6.7 The information content of removal notification includes (see clause 7.38):

a) the BioAPI Unit ID;

b) event type (remove);

c) callback context.

6.6.8 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***.

6.7 Controlling BioAPI Units

6.7.1 A function (see clauses 8.1.13 and 9.3.1.8) 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 7.22 (BioAPI 2.0), 7.23 (BioAPI 2.1 or greater), and 7.70).

NOTE The schema may not be available in framework-free systems.

6.7.2 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 document.

6.8 BIR Structure and Handling

6.8.1 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 7.7.

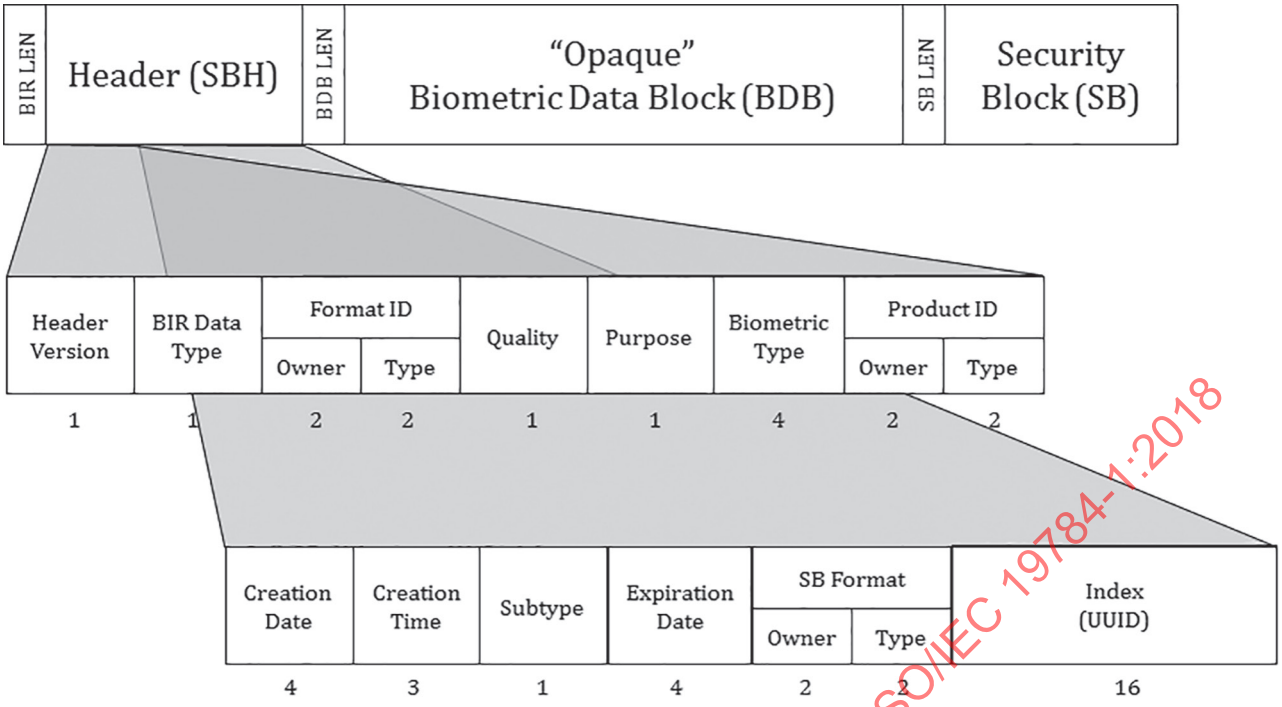

Figure 3 — Biometric Information Record (serialized BIR for storage and transmission)

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 7.13).

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).

6.8.2 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 ISO/IEC 24708[6]), it can acquire the BIR using the handle (in the function ***BioAPI_GetBIRFromHandle***). If just the header information is needed, it can be retrieved using ***BioAPI_GetHeaderFromHandle***.

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

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

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

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

7 BioAPI types and macros

NOTE Many of these types and macros are not relevant for the implementation of a BSP designed to work in a framework-free BioAPI system.

7.1 BioAPI

Definition of the BioAPI calling conventions.

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

Note that 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_ACBio_PARAMETERS (BioAPI 2.2)

7.2.1 Structure giving information which is used to generate ACBio instances

```
typedef struct bioapi_acbio_parameters {
   uint32_t Challenge[4];
   uint32_t *InitialBPUIOIndexOutput;
   uint32_t *SupremumBPUIOIndexOutput;
} BioAPI_ACBio_PARAMETERS;
```

7.2.2 Definitions

— *Challenge* – Challenge from the validator of a biometric verificatioin when ACBio is used. This value shall be set to the field `controlValue` of type `ACBioContentInformation` in ACBio instances.

— *InitialBPUIOIndexOutput* – The initial value of BPU IO index which is to be assigned to the output from the BioAPI Unit, BFP, or BSP when the ACBio instances are generated. The range between *InitialBPUIOIndexOutput* and *SupremumBPUIOIndexOutput* shall be divided into the number of BSP Units and BFPs which are attached to the BSP, and assigned to the BSP Units and BSPs.

— *SupremumBPUIOIndexOutput* – The supremum of BPU IO indexes which are to be assigned to the output from the BioAPI Unit, BFP, or BSP when the ACBio instances are generated.

7.3 BioAPI_ASN1_BIR (BioAPI 2.2)

A container for biometric data, (or non-biometric data) that may affect a biometric operation, encoded in ASN.1 DER. The BioAPI_ASN1_BIR structure is used when a BioAPI API with security functionality is used. The BioAPI_ASN1_BIR structure associates a length, in bytes, with the address of an arbitrary block of contiguous memory which contains an ASN.1 encoded BIR of type `BioAPI-BIR`, specified in Annex E. The ASN.1 encoded BIR consists of an SBH of type `BioAPIBIRHeader`, a BDB of type `BiometricData`, and an SB of type `CBEFFSecurityBlock`. The BDB may contain raw sample data, partially processed (intermediate) data, completely processed data, score data (resulting from a matching or fusion operation), decision data (resulting from a decision or fusion operation), or biographic data which can be provided as input to a biometric operation to modify its behavior. The BioAPI_ASN1_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). It may also be stored and used later to provide feedback to subsequent biometric operations. Type BioAPI_ASN1_BIR is an alias of type BioAPI_DATA.

NOTE The ASN.1 type BioAPI-BIR corresponds to the C structured type BioAPI_BIR element by element.

```
typedef BioAPI_DATA BioAPI_ASN1_BIR;
```

7.4 BioAPI_ASN1_ENCODED (BioAPI 2.2)

A container for ASN.1 DER encoded data. The BioAPI_ASN1_ENCODED structure is used to express information about cryptographic keys. The BioAPI _ASN1_ ENCODED structure associates a length, in bytes, with the address of an arbitrary block of contiguous memory which contains an ASN.1 encoded data. Type BioAPI_ASN1_ENCODED is an alias of type BioAPI_DATA.

typedef BioAPI_DATA BioAPI_ASN1_ENCODED;

7.5 BioAPI_BFP_LIST_ELEMENT

7.5.1 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;
```

7.5.2 Definitions

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

— *BFPUuid* – UUID of the BFP in the component registry.

7.6 BioAPI_BFP_SCHEMA

7.6.1 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;
```

7.6.2 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 10646:2017, Annex D).

NOTE 1 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 2 FPI specifications will be issued as subsequent, additional parts of the ISO/IEC 19784 series.

— *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.

7.7 BioAPI_BIR

7.7.1 A container for biometric data, or non-biometric data that may affect a biometric operation. 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, completely processed data, score data (resulting from a matching or fusion operation), decision data (resulting from a decision or fusion operation), or biographic data which can be provided as input to a biometric operation to modify its behavior. 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). It may also be stored and used later to provide feedback to subsequent biometric operations.

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

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

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

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

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

7.8 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;
```

7.9 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. 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.

7.10 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. 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.

7.11 BioAPI_BIR_BIOMETRIC_TYPE (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

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

```
typedef uint32_t BioAPI_BIR_BIOMETRIC_TYPE;

   #define BioAPI_NO_TYPE_AVAILABLE        (0x00000000)
   #define BioAPI_TYPE_MULTIPLE            (0x00000001)
   #define BioAPI_TYPE_FACIAL_FEATURES     (0x00000002)
   #define BioAPI_TYPE_VOICE               (0x00000004)
   #define BioAPI_TYPE_FINGERPRINT         (0x00000008)
   #define BioAPI_TYPE_IRIS                (0x00000010)
   #define BioAPI_TYPE_RETINA              (0x00000020)
   #define BioAPI_TYPE_HAND_GEOMETRY       (0x00000040)
```



```
#define BioAPI_TYPE_SIGNATURE_DYNAMICS (0x00000080)
#define BioAPI_TYPE_KEYSTOKE_DYNAMICS  (0x00000100)
#define BioAPI_TYPE_LIP_MOVEMENT       (0x00000200)
#define BioAPI_TYPE_THERMAL_FACE_IMAGE (0x00000400)
#define BioAPI_TYPE_THERMAL_HAND_IMAGE (0x00000800)
#define BioAPI_TYPE_GAIT               (0x00001000)
#define BioAPI_TYPE_OTHER              (0x40000000)
#define BioAPI_TYPE_PASSWORD           (0x80000000)
```

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

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

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

7.12 BioAPI_BIR_BIOMETRIC_TYPE (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

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

In the following definition of BioAPI_BIR_BIOMETRIC_TYPE, 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.

```
typedef uint32_t BioAPI_BIR_BIOMETRIC_TYPE;

#define BioAPI_NO_BIOTYPE_AVAILABLE           (0x00000000)
#define BioAPI_TYPE_MULTIPLE_BIOMETRIC_TYPES  (0x00000001)
#define BioAPI_TYPE_FACE                      (0x00000002)
#define BioAPI_TYPE_VOICE                     (0x00000004)
#define BioAPI_TYPE_FINGER                    (0x00000008)
#define BioAPI_TYPE_IRIS                      (0x00000010)
#define BioAPI_TYPE_RETINA                    (0x00000020)
#define BioAPI_TYPE_HAND_GEOMETRY             (0x00000040)
#define BioAPI_TYPE_SIGNATURE_SIGN            (0x00000080)
#define BioAPI_TYPE_KEYSTROKE                 (0x00000100)
#define BioAPI_TYPE_LIP_MOVEMENT              (0x00000200)
#define BioAPI_TYPE_GAIT                      (0x00001000)
#define BioAPI_TYPE_VEIN                      (0x00002000)
#define BioAPI_TYPE_DNA                       (0x00004000)
#define BioAPI_TYPE_EAR                       (0x00008000)
#define BioAPI_TYPE_FOOT                      (0x00010000)
#define BioAPI_TYPE_SCENT                     (0x00020000)
#define BioAPI_TYPE_OTHER                     (0x40000000)
#define BioAPI_TYPE_PASSWORD                  (0x80000000)
```

NOTE 1 BioAPI_TYPE_MULTIPLE_BIOMETRIC_TYPES 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 BioAPI BIR Biometric Type corresponds to the "CBEFF_BDB_biometric_type" in ISO/IEC 19785-1.

NOTE 3 Although “password” is not a biometric characteristic, BioAPI_TYPE_PASSWORD is included as a valid BioAPI_BIR_BIOMETRIC_TYPE to support its use a) for development & test environments, and b) as an additional authentication factor within a BioAPI application.

NOTE 4 The names of several biometric types are different in BioAPI 2.1 or greater from the names used in BioAPI 2.0, with no change in semantics, and several new biometric types are present in BioAPI 2.1 or greater. Finally, the "thermal" types are present in BioAPI 2.0 but are absent from BioAPI 2.1 or greater. These differences are in order to align with ISO/IEC 19785-1.

NOTE 5 The names of the following values differ between BioAPI 2.0 and BioAPI 2.1 or greater, but their semantics is the same. The BioAPI 2.1 or greater names are aligned with ISO/IEC 19785-1 (CBEFF).

Name in BioAPI 2.0	Name in BioAPI 2.1 or greater	Encoding
MULTIPLE	MULTIPLE_BIOMETRIC_TYPES	0x00000001
FACIAL_FEATURES	FACE	0x00000002
FINGERPRINT	FINGER	0x00000008
SIGNATURE_DYNAMICS	SIGNATURE_SIGN	0x00000080
KEYSTROKE_DYNAMICS	KEYSTROKE	0x00000100

NOTE 6 The following values are not present BioAPI 2.0 but are present in BioAPI 2.1 or greater for consistency with ISO/IEC 19785-1 (CBEFF).

Added value	Encoding
NO_BIOTYPE_AVAILABLE	0x00000000
VEIN	0x00002000
DNA	0x00004000
EAR	0x00008000
FOOT	0x00010000
SCENT	0x00020000

7.13 BioAPI_BIR_DATA_TYPE

7.13.1 This data type is used for three different purposes:

a) it identifies the type of biometric sample (raw, intermediate, processed, score data, decision data or biographic data) 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.

7.13.2 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.)

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

7.13.4 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_SCORE            (0x08)
   #define BioAPI_BIR_DATA_TYPE_DECISION         (0x09)
   #define BioAPI_BIR_DATA_TYPE_BIOGRAPHIC       (0x0A)
   #define BioAPI_BIR_DATA_TYPE_ENCRYPTED        (0x10)
   #define BioAPI_BIR_DATA_TYPE_SIGNED           (0x20)
   #defineBioAPI_BIR_INDEX_PRESENT               (0x80)
```

NOTE 1 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.

NOTE 2 BioAPI_BIR_DATA_TYPE_DECISION (BioAPI 2.2 or greater) and BioAPI_BIR_DATA_TYPE_BIOGRAPHIC (BioAPI 2.2 or greater) have two bits on while others have only one bit on.

NOTE 3 BioAPI_BIR_DATA_TYPE_SCORE is used in BioAPI 2.2 or greater.

7.14 BioAPI_BIR_HANDLE

A handle to refer to a BioAPI BIR or an ASN.1 encoded 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)
```

7.15 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.

7.16 BioAPI_BIR_PURPOSE

7.16.1 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_PURPOSE_DECIDE                              (7)
   #define BioAPI_NO_PURPOSE_AVAILABLE                        (0)
```

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.

NOTE BioAPI_PURPOSE_DECIDE is used in BioAPI 2.2 or greater.

7.16.2 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.

7.16.3 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***, ***BioAPI_ProcessWithAuxBIR*** (BioAPI 2.1 or less), ***BioAPI_ProcessUsingAuxBIRs*** (BioAPI 2.2 or greater), ***BioAPI_Decide*** (BioAPI 2.2 or greater)***, and BioAPI_Fuse*** (BioAPI 2.2 or greater) 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*** and ***BioAPI_ProcessUsingAuxBIRs*** (BioAPI 2.2 or greater) function may accept as input any intermediate BIR with a Purpose of BioAPI_PURPOSE_VERIFY or BioAPI_PURPOSE_IDENTIFY, and shall output only BIRs with the same purpose as the input BIR.

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

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

i) All score BIRs and decision BIRs must have the Decide purpose. Biographic BIRs may have any purpose. No other types of BIRs may have the Decide purpose.

7.17 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' flag 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. 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 can 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.

7.18 BioAPI_BIR_SUBTYPE (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

7.18.1 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.

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

7.18.3 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.

7.19 BioAPI_BIR_SUBTYPE (BioAPI 2.1)

7.19.1 This subclause applies only when the BioAPI version number in use is 2.1 or greater.

7.19.2 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.

NOTE In the BioAPI 2.1 definition of this type, multiple bits can be set. This definition is aligned with ISO/IEC 19785-1.

```
typedef uint8_t BioAPI_BIR_SUBTYPE;

   #define BioAPI_BIR_SUBTYPE_VEIN_ONLY_MASK   (0x80)
   #define BioAPI_BIR_SUBTYPE_LEFT_MASK        (0x01)
   #define BioAPI_BIR_SUBTYPE_RIGHT_MASK       (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_VEIN_PALM        (0x04)
   #define BioAPI_BIR_SUBTYPE_VEIN_BACKOFHAND  (0x08)
   #define BioAPI_BIR_SUBTYPE_VEIN_WRIST       (0x10)
   #define BioAPI_NO_SUBTYPE_AVAILABLE         (0x00)
```

7.19.3 This structure is a bitmask, with the bits defined as shown below. Bit 7 is used to indicate the interpretation of the lower-order bits. Bit positions zero (0) and one (1) always indicate left and right respectively. When bit position 7 is not set, these bit positions may apply to any biometric type; however, bit positions 2-6 are specific to the finger and vein biometric types. When bit position 7 is set, then the remaining bit positions apply to the vein biometric type only.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (Any)	Little	Ring	Middle	Pointer	Thumb	Right	Left
1 (Vein only)	Reserved	Reserved	Wrist	Back of Hand	Palm	Right	Left

NOTE Vein biometric data can be obtained from the fingers (Bit 7 set to zero), or from the wrists, palm, or backs of either hand (Bit 7 set to 1).

7.19.4 Zero or more bits may be set. The abstract value NO VALUE AVAILABLE is indicated by setting all bits to zero.

NOTE The abstract value NO VALUE AVAILABLE can be used with BIRs that are not originally created by a BioAPI BSP but have been transformed from another data format. It can also be used when none of the other values are applicable or information is not available.

7.19.5 The bit positions BioAPI_BIR_SUBTYPE_THUMB through BioAPI_BIR_SUBTYPE_LITTLEFINGER can be used to identify the instance(s) of a biometric type related to the fingers.

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

7.19.6 If one or more of the bit positions BioAPI_BIR_SUBTYPE_THUMB through BioAPI_BIR_SUBTYPE_LITTLEFINGER are set, then either or both the bit positions BioAPI_BIR_SUBTYPE_LEFT_MASK and BioAPI_BIR_SUBTYPE_RIGHT_MASK shall also be set. When only the former is set, the subtype value designates one or more fingers of the left hand. When only the latter is set, the subtype value designates one or more fingers of the right hand. When both are set, the value designates one or more fingers of both hands (the same finger locations in both).

NOTE It is not possible to designate, for example, a set of fingers consisting of the index finger of the left hand and the medium finger of the right hand.

7.19.7 When a value with multiple finger bits set occurs in a BIR header (or is used as an input parameter of a BioAPI function that performs capture), it is not strictly required that all the specified finger instances be actually present in the BDB of the BIR (or be actually captured). However, it is recommended that all the designated finger instances be present in the BDB except in the case of missing fingers and similar exceptional situations. Likewise, if the subtype field of a BIR designates multiple fingers, it is acceptable for the BDB of that BIR to include data about an extra finger if the subject happens to have six fingers in his or her hand.

7.19.8 If none of the finger bit positions BioAPI_BIR_SUBTYPE_THUMB through BioAPI_BIR_SUBTYPE_LITTLEFINGER are set, then the bit positions BioAPI_BIR_SUBTYPE_LEFT_MASK and BioAPI_BIR_SUBTYPE_RIGHT_MASK can be used to identify an instance of a biometric type for which there is one left instance and one right instance (for example, iris), or also the single instance of a biometric type for which there is only one instance (for example, face).

7.19.9 Vein data taken from one or more fingers is indicated by setting bit position 7 to zero, setting one or both of bit positions 0 and 1, and setting one or more of the finger bit positions (3-7). Vein data taken from one of the other sources (wrist, palm, or back of hand) is indicated by setting bit position 7 to one, setting one or both of bit positions 0 and 1, and setting one of bit positions 3-5 as appropriate).

NOTE 1 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 can use the NO VALUE AVAILABLE condition.

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 subtype 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.

7.19.10 The use of the bit position BioAPI_BIR_SUBTYPE_MULTIPLE is not recommended when the BioAPI version number in use is 2.1 or greater.

7.20 BioAPI_BIR_SUBTYPE_MASK (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

The type BioAPI_BIR_SUBTYPE_MASK is defined as follows.

```
typedef uint32_t BioAPI_BIR_SUBTYPE_MASK;

      #define BioAPI_BIR_SUBTYPE_LEFT_BIT                  (0x0001)
      #define BioAPI_BIR_SUBTYPE_RIGHT_BIT                 (0x0002)
      #define BioAPI_BIR_SUBTYPE_LEFT_THUMB_BIT            (0x0004)
      #define BioAPI_BIR_SUBTYPE_LEFT_POINTERFINGER_BIT    (0x0008)
      #define BioAPI_BIR_SUBTYPE_LEFT_MIDDLEFINGER_BIT     (0x0010)
      #define BioAPI_BIR_SUBTYPE_LEFT_RINGFINGER_BIT       (0x0020)
      #define BioAPI_BIR_SUBTYPE_LEFT_LITTLEFINGER_BIT     (0x0040)
      #define BioAPI_BIR_SUBTYPE_RIGHT_THUMB_BIT           (0x0080)
      #define BioAPI_BIR_SUBTYPE_RIGHT_POINTERFINGER_BIT   (0x0100)
      #define BioAPI_BIR_SUBTYPE_RIGHT_MIDDLEFINGER_BIT    (0x0200)
      #define BioAPI_BIR_SUBTYPE_RIGHT_RINGFINGER_BIT      (0x0400)
      #define BioAPI_BIR_SUBTYPE_RIGHT_LITTLEFINGER_BIT    (0x0800)
      #define BioAPI_BIR_SUBTYPE_LEFT_VEIN_PALM_BIT        (0x00001000)
      #define BioAPI_BIR_SUBTYPE_LEFT_VEIN_BACKOFHAND_BIT  (0x00002000)
      #define BioAPI_BIR_SUBTYPE_LEFT_VEIN_WRIST_BIT       (0x00004000)
      #define BioAPI_BIR_SUBTYPE_RIGHT_VEIN_PALM_BIT       (0x00008000)
      #define BioAPI_BIR_SUBTYPE_RIGHT_VEIN_BACKOFHAND_BIT (0x00010000)
      #define BioAPI_BIR_SUBTYPE_RIGHT_VEIN_WRIST_BIT      (0x00020000)
```

NOTE 1 The VEIN values are not present in BioAPI 2.0.

The bit positions BioAPI_BIR_SUBTYPE_LEFT_BIT and BioAPI_BIR_SUBTYPE_RIGHT_BIT can be used to identify the instance of a biometric modality for which there is one "left" instance and one "right" instance.

NOTE 2 Iris and hand geometry are examples of such modalities.

The other bit positions (from BioAPI_BIR_SUBTYPE_LEFT_THUMB_BIT through BioAPI_BIR_SUBTYPE_RIGHT_LITTLEFINGER_BIT) can be used to identify the instance of a biometric modality related to the fingers.

NOTE 3 Fingerprint is one such modality.

The value zero (no bit positions set) can be used with biometric modalities for which there is only one instance.

NOTE 4 Signature/sign is one such modality.

7.21 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)
```

7.22 BioAPI_BSP_SCHEMA (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

7.22.1 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;
```

7.22.2 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 10646:2017 Annex D).

NOTE 1 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 2 Applies only when a BSP is only capable of directly managing a single archive unit.

NOTE 3 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.

7.22.3 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.

7.23 BioAPI_BSP_SCHEMA (BioAPI 2.1)

7.23.1 This subclause applies only when the BioAPI version number in use is 2.1 or greater.

7.23.2 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;
   uint32_t MaxNumEnrollInstances;
   uint8_t  *HostingEndpointIRI;
   BioAPI_UUID BSPAccessUUID;
}BioAPI_BSP_SCHEMA;
```

7.23.3 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 10646:2017, 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 2 Applies only when a BSP is only capable of directly managing a single archive unit.

NOTE 3 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.

— *MaxNumEnrollInstances* – The maximum number of distinct instances that a BSP can create reference templates for in one enroll operation. This information can be useful to an application that uses the application-controlled GUI feature.

— *HostingEndpointIRI* – An IRI identifying the framework whose component registry contains a registration of the BSP. This parameter shall be ignored by frameworks conforming to this document and shall be set to NULL by an application. It is provided to support interworking standards, which may specify the use of identical BSPs present on multiple computers from within an application running on the same or a different computer.

— *BSPAccessUUID* – A UUID, unique within the scope of an application, which the application may use to refer to the BSP as an alternative to the BSP product UUID. This parameter shall be ignored by frameworks conforming to this document and can be set to any UUID value by an application. It is provided to support interworking standards, which may specify the use of identical BSPs present on multiple computers from within an application running on the same or a different computer.

NOTE 4 The "BSPAccess_UUID" and the "HostingendpointIRI" are part of the definition of the C type **BioAPI_BSP_SCHEMA**, but are not part of the BSP schema information stored in the component registry (see Table 3).

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

7.24 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;
```

7.25 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.

7.26 BioAPI_DATA

7.26.1 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;
```

7.26.2 Definitions

— *Length* – Length of the data buffer in bytes.

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

7.27 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].

7.28 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)
```

7.29 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;
```

7.30 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)
```

7.31 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.

7.32 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.

7.33 BioAPI_ENCRYPTION_ALG (BioAPI 2.2)

Identifies encryption algorithm supported by a BioAPI Unit. This BioAPI type shall be the XML value notation of ASN.1 identifier (see ISO/IEC 8824-1) assigned to encryption algorithms. Example to specify AES with 128 bit keys in CBC mode, the char would be "2.16.840.1.101.3.4.1.2" which is the XML value notation for.

```
typedef char *BioAPI_ENCRYPTION_ALG;

   #define BioAPI_ENCRYPTION_ALG_NOT_SUPPORTED NULL
```

7.34 BioAPI_ENCRYPTION_INFO (BioAPI 2.2)

7.34.1 Structure giving information of the cryptographic algorithm and key(s) of a BioAPI Unit or a biometric application which is used to encrypt/decrypt biometric data

```
typedef struct bioapi_encryption_info {
   BioAPI_ENCRYPTION_ALG ENCAlg;
   BioAPI_KEY_INFO *ENCKeyInfo;
   uint32_t *NumberOfElementsEncKeyInfo;
} BioAPI_ENCRYPTION_INFO;
```

7.34.2 Definitions

— *ENCAlg* – Encryption algorithm.

— *ENCKeyInfo* – An array of key information for encryption.

— *NumberOfElementsEncKeyInfo* – The number of elements in the array of EncKeyInfo.

7.35 BioAPI_ERROR_INFO (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

This structure is used by the function ***BioAPI_GetLastErrorInfo***, and holds information about the most recent error condition that caused a BioAPI function call to return an error code.

```
typedef struct bioapi_error_info {
   int32_t ErrorCode;
   BioAPI_STRING ErrorMessage;
   BioAPI_STRING ErrorSourceName;
} BioAPI_ERROR_INFO;
```

The member ErrorCode is the same value that was returned by the last BioAPI function call that returned an error code. The value of the remaining members are not standardized.

7.36 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)
```

7.37 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)
```

When the BioAPI version number in use is 2.0, it can be impossible to mask an INSERT event coming from an attach session of a BSP, because the event can 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*/*BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater), and ***BioAPI_BSPAttach*/*BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) 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***.

7.38 BioAPI_EventHandler

7.38.1 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);
```

7.38.2 Definitions

— *BSPUuid (input) – The UUID of the BSP raising the event.*

— *UnitID (input) – 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 (input) – A pointer to the unit schema of the BioAPI Unit associated with the event.*

— *EventType (input) – 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.

7.39 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).

7.40 BioAPI_FRAMEWORK_SCHEMA

7.40.1 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;
```

7.40.2 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 10646:2017, 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.

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

7.41 BioAPI_GUI_BITMAP (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

7.41.1 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;
```

7.41.2 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 7.54 and 7.55.

7.42 BioAPI_GUI_BITMAP (BioAPI 2.1)

7.42.1 This subclause applies only when the BioAPI version number in use is 2.1 or greater.

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

```
typedef struct bioapi_gui_bitmap {
   BioAPI_BIR_SUBTYPE_MASK SubtypeMask;
       /* The subtype (or subtypes) of the sample represented
       by the 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;
```

7.42.3 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_EVENT_HANDLER and BioAPI_GUI_PROGRESS_EVENT_HANDLER descriptions in 7.86.

7.43 BioAPI_GUI_BITMAP_ARRAY (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

The type BioAPI_GUI_BITMAP_ARRAY is defined as follows:

```
typedef struct bioapi_gui_bitmap_array {
   uint32_t           NumberOfMembers;
   BioAPI_GUI_BITMAP      *Bitmaps;
       /* A pointer to an array of BioAPI_GUI_BITMAPs */
} BioAPI_GUI_BITMAP_ARRAY;
```

7.44 BioAPI_GUI_ENROLL_TYPE (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

The type BioAPI_GUI_ENROLL_TYPE indicates the enroll type of a BSP, and is defined as follows:

```
typedef uint32_t BioAPI_GUI_ENROLL_TYPE;

   #define BioAPI_GUI_ENROLL_TYPE_TEST_VERIFY      (0x00000001)
   #define BioAPI_GUI_ENROLL_TYPE_MULTIPLE_CAPTURE (0x00000002)
```

The enroll type of a BSP denotes a pattern of suboperations performed by the BSP during an enroll operation.

The bit position BioAPI_GUI_ENROLL_TYPE_TEST_VERIFY indicates, when set, that the BSP supports test verification at enrollment. During an enroll operation, an application receives GUI state event notification callbacks for two capture suboperations, GUI state event notification callbacks for a process suboperation, GUI state event notification callbacks for a create template suboperation, and GUI state event notification callbacks for a verify match suboperation;

The bit position BioAPI_GUI_ENROLL_TYPE_MULTIPLE_CAPTURE indicates, when set, that the BSP supports multiple capture suboperations at enrollment. During an enroll operation, an application

receives GUI state event notification callbacks for multiple capture suboperations followed by GUI state event notification callbacks for a create template suboperation.

The two bit positions shall not be set at the same time.

NOTE This restriction is imposed because an application, in order to support a combination of Test-Verify and Multiple-Capture, would need more information about the timing of the suboperations in order to decide the layout of its screen during enrollment.

7.45 BioAPI_GUI_EVENT_SUBSCRIPTION (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

This type carries information about an existing named GUI event subscription. It identifies the subscriber application and the named subscription, and indicates which types of GUI events are in the scope of the subscription. This type is specified for use in the BioAPI_QueryGUIEventSubscriptions function.

A named GUI event subscription is one that was created by a call to BioAPI_SubscribeToGUIEvents specifying a non-NULL GUI event subscription UUID. The framework calls the event handlers specified in a named subscription to notify BSP-generated GUI events that have been redirected to that named subscription (see 8.3.7), and to notify application-generated GUI events (see 8.3.3, 8.3.4, and 8.3.5) directed to that named subscription.

```
typedef struct _bioapi_gui_event_subscription  {
   const uint8_t *SubscriberEndpointIRI;
   BioAPI_UUID GUIEventSubscriptionUuid;
   BioAPI_BOOL GUISelectEventSubscribed;
   BioAPI_BOOL GUIStateEventSubscribed;
   BioAPI_BOOL GUIProgressEventSubscribed;
} BioAPI_GUI_EVENT_SUBSCRIPTION;
```

— *SubscriberEndpointIRI* – An IRI -see RFC 3987- (originally provided by the framework) that identifies the application that created the named GUI event subscription. This shall be NULL if the subscriber application is the same as the current application.

— `GUIEventSubscriptionUuid` – A UUID (originally provided by the subscriber application) that identifies the named GUI event subscription.

— `GUISelectEventSubscribed` – Indicates whether GUI select events are in the scope of the subscription (a non-`NULL` callback address was originally provided by the subscriber application for the GUI select event handler).

— `GUIStateEventSubscribed` – Indicates whether GUI state events are in the scope of the subscription (a non-`NULL` callback address was originally provided by the subscriber application for the GUI state event handler).

— `GUIProgressEventSubscribed` – Indicates whether GUI progress events are in the scope of the subscription (a non-`NULL` callback address was originally provided by the subscriber application for the GUI progress event handler).

7.46 BioAPI_GUI_MESSAGE (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

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 7.41.

7.47 BioAPI_GUI_MOMENT (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

An enumeration of the moments in which:

a) a GUI select event can be generated with respect to a suboperation cycle, or

b) a GUI state or progress event can be generated with respect to a suboperation.

```
typedef uint32_t BioAPI_GUI_MOMENT;

   #define BioAPI_GUI_MOMENT_BEFORE_START (1)
   #define BioAPI_GUI_MOMENT_DURING (2)
   #define BioAPI_GUI_MOMENT_AFTER_END (3)
```

The values BioAPI_GUI_MOMENT_BEFORE_START and BioAPI_GUI_MOMENT_AFTER_END can occur in all types of GUI event notifications, whereas the value BioAPI_GUI_MOMENT_DURING can only occur in GUI progress event notifications.

When the value BioAPI_GUI_MOMENT_BEFORE_START occurs in a GUI select event notification, it indicates that the BSP is ready to start a new suboperation cycle. A suboperation cycle consists of a single execution of the sequence of suboperations that comprise an operation (see 7.82). There is no limit to the number of suboperation cycles (successful or not) that the application can request to be performed for an operation. However, all the data produced in each cycle shall be discarded when a subsequent cycle is started, and therefore the outcome of an operation (including the return value and the data produced) is always the outcome of its last (or only) suboperation cycle (except in the case of cancellation).

When the value BioAPI_GUI_MOMENT_AFTER_END occurs in a GUI select event notification callback, it indicates that the current suboperation cycle has ended. This value does not indicate whether the suboperation cycle has been successful or not, as that information is provided by a separate parameter of the GUI select event notification that contains the result value of the suboperation cycle.

When the value BioAPI_GUI_MOMENT_BEFORE_START occurs in a GUI state event notification callback, it indicates that the BSP is ready to start a new suboperation within the current suboperation cycle.

When the value BioAPI_GUI_MOMENT_AFTER_END occurs in a GUI state event notification callback, it indicates that the current suboperation performmed by the BSP has ended. This value does not indicate whether the suboperation has been successful or not, as that information is provided by a separate parameter of the GUI state event notification which carries the result value of the suboperation cycle.

When the value BioAPI_GUI_MOMENT_BEFORE_START occurs in a GUI progress event notification callback, it indicates that the BSP is ready to start a new suboperation within the current suboperation cycle. A GUI progress event with this moment value may be redundant when a GUI state event is also produced, and may be omitted. However, such a GUI progress event is useful for those operations (such as the identify match operation) that do not produce GUI state events.

When the value BioAPI_GUI_MOMENT_DURING occurs in a GUI progress event notification callback, it indicates that the suboperation is in progress.

When the value BioAPI_GUI_MOMENT_AFTER_END occurs in a GUI progress event notification callback, it indicates that the current suboperation performmed by the BSP has ended. A GUI progress event with this moment value may be redundant when a GUI state event is also produced, and may be omitted. However, such a GUI progress event is useful for those operations (such as the identify match operation) that do not produce GUI state events.

NOTE This type is used with the application-controlled GUI feature (see 7.86).

7.48 BioAPI_GUI_OPERATION (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

An enumeration of all BioSPI functions (operations) during which a BSP can generate GUI events.

```
typedef uint8_t BioAPI_GUI_OPERATION;

   #define BioAPI_GUI_OPERATION_CAPTURE (1)
   #define BioAPI_GUI_OPERATION_PROCESS (2)
   #define BioAPI_GUI_OPERATION_CREATETEMPLATE (3)
   #define BioAPI_GUI_OPERATION_VERIFYMATCH (4)
   #define BioAPI_GUI_OPERATION_IDENTIFYMATCH (5)
   #define BioAPI_GUI_OPERATION_VERIFY (6)
   #define BioAPI_GUI_OPERATION_IDENTIFY (7)
   #define BioAPI_GUI_OPERATION_ENROLL (8)
```

The values of this type occur in GUI event notifications, and identify the type of BioSPI function that was in execution when the BSP produced the GUI event, and to which the GUI event refers.

The following table specifies the correspondence between BioSPI functions and operations.

Table 2 — BioSPI functions and operations

BioSPI function	`BioAPI_GUI_OPERATION` value	Common name of the operation
BioSPI_Capture	BioAPI_GUI_OPERATION_CAPTURE	capture operation
BioSPI_Process	BioAPI_GUI_OPERATION_PROCESS	process operation
BioSPI_CreateTemplate	BioAPI_GUI_OPERATION_CREATETEMPLATE	create template operation
BioSPI_VerifyMatch	BioAPI_GUI_OPERATION_VERIFYMATCH	verify match operation
BioSPI_IdentifyMatch	BioAPI_GUI_OPERATION_IDENTIFYMATCH	identify match operation
BioSPI_Verify	BioAPI_GUI_OPERATION_VERIFY	verify operation
BioSPI_Identify	BioAPI_GUI_OPERATION_IDENTIFY	identify operation
BioSPI_Enroll	BioAPI_GUI_OPERATION_ENROLL	enroll operation

The following table specifies which GUI events can be generated by a BSP during the execution of each type of operation.

Table 3 — GUI events generated in each operation

Operation	GUI select events	GUI state events	GUI progress events
capture operation	X	X	X
process operation			X
Create template operation			X
verify match operation			X
identify match operation			X
verify operation	X	X	X
identify operation	X	X	X
enroll operation	X	X	X

7.49 BioAPI_GUI_PROGRESS (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

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 7.54.

7.50 BioAPI_GUI_PROGRESS (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

A value in the range 0 to 100 indicating the percent of completion of a suboperation, intended for display to the subject or an operator.

```
typedef uint8_t BioAPI_GUI_PROGRESS;
```

NOTE This is used with the application-controlled GUI option. See BioAPI_GUI_PROGRESS_EVENT description in 7.86.

7.51 BioAPI_GUI_RESPONSE (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

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 7.54.

7.52 BioAPI_GUI_RESPONSE (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

An enumeration of the possible actions to be performed by a BSP after a GUI event notification callback made by the BSP has returned control to the BSP. Before returning from a notification callback, an application assigns a value of this type to an output parameter of the callback (`Response`).

```
typedef uint8_t BioAPI_GUI_RESPONSE;

   #define BioAPI_GUI_RESPONSE_DEFAULT           (0)
   #define BioAPI_GUI_RESPONSE_OP_COMPLETE       (1)
   #define BioAPI_GUI_RESPONSE_OP_CANCEL         (2)
   #define BioAPI_GUI_RESPONSE_CYCLE_START       (3)
   #define BioAPI_GUI_RESPONSE_CYCLE_RESTART     (4)
   #define BioAPI_GUI_RESPONSE_SUBOP_START       (5)
   #define BioAPI_GUI_RESPONSE_SUBOP_NEXT        (6)
   #define BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE (7)
   #define BioAPI_GUI_RESPONSE_PROGRESS_ABORT    (8)
   #define BioAPI_GUI_RESPONSE_RECAPTURE         (9)
```

The value BioAPI_GUI_RESPONSE_OP_COMPLETE can only be returned in response to a GUI select event notification callback with the moment value BioAPI_GUI_MOMENT_AFTER_END. It indicates that

the BSP should consider the entire operation complete and should cause the original function call to return the result code of the current suboperation cycle (which can be either BioAPI_OK or an error code). After this response from the application, no further GUI event notification callbacks are expected for the same operation.

The value BioAPI_GUI_RESPONSE_OP_CANCEL can be returned in response to any GUI event notification callback. It indicates that the BSP should abort the entire operation and cause the original function call to return an error code. After this response from the application, no further GUI event notification callbacks are expected from the BSP for the same operation.

The value BioAPI_GUI_RESPONSE_CYCLE_START can only be returned in response to a GUI select event notification callback with the moment value BioAPI_GUI_MOMENT_BEFORE_START. It indicates that the BSP should start the suboperation cycle.

The value BioAPI_GUI_RESPONSE_CYCLE_RESTART can only be returned in response to a GUI select event notification callback with the moment value BioAPI_GUI_MOMENT_AFTER_END or in response to a GUI state or progress event notification callback with any moment value. It indicates that the BSP should discard all the information produced during the current suboperation cycle and should start a new suboperation cycle. After this response from the application, a GUI select event notification callback with the moment value BioAPI_GUI_MOMENT_BEFORE_START is expected from the BSP.

The value BioAPI_GUI_RESPONSE_SUBOP_START can only be returned in response to a GUI state event notification callback with the moment value BioAPI_GUI_MOMENT_BEFORE_START. It indicates that the BSP should start the suboperation.

The value BioAPI_GUI_RESPONSE_SUBOP_NEXT can only be returned in response to a GUI state event notification callback with the moment value BioAPI_GUI_MOMENT_AFTER_END. It indicates that the BSP should either perform the next suboperation in the current suboperation cycle (if there are more suboperations to be performed) or exit the suboperation cycle (if all the suboperations in the cycle have been performed). After this response from the application, either a GUI state event notification callback with the moment value BioAPI_GUI_MOMENT_BEFORE_START or a GUI select event notification callback with the moment value BioAPI_GUI_MOMENT_AFTER_END is expected from the BSP (respectively).

The value BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE can only be returned in response to a GUI progress event notification callback. It indicates that the BSP should continue performing the suboperation.

The value BioAPI_GUI_RESPONSE_PROGRESS_ABORT can only be returned in response to a GUI progress event notification callback. It indicates that the BSP should abort the suboperation and produce an error code. After this response from the application, a GUI state event notification callback with the moment value BioAPI_GUI_MOMENT_AFTER_END is expected from the BSP.

The value BioAPI_GUI_RESPONSE_RECAPTURE can only be returned in response to a GUI state event notification callback with the moment value BioAPI_GUI_MOMENT_AFTER_END. It indicates that the BSP should discard one of the samples captured in the current suboperation cycle (possibly also discarding any reference template and any processed sample created from the sample being discarded) and perform another capture suboperation to produce a new sample. After this response from the application, a GUI state event notification callback with the moment value BioAPI_GUI_MOMENT_BEFORE_START (for a capture suboperation with the purpose of enrollment) is expected from the BSP. When the capture suboperation is completed, the BSP shall perform any other suboperations required to complete the current suboperation cycle.

The value BioAPI_GUI_RESPONSE_DEFAULT shall be interpreted as the one among the following values which applies to the specific situation in which it is returned: BioAPI_GUI_RESPONSE_CYCLE_START, BioAPI_GUI_RESPONSE_SUBOP_START, BioAPI_GUI_RESPONSE_PROCESS_CONTINUE, BioAPI_GUI_RESPONSE_SUBOP_NEXT, or BioAPI_GUI_RESPONSE_OP_COMPLETE.

If an application returns a response value other than those permitted in each situation, the BSP shall abort the entire operation and cause the original function call to return an error code. No further GUI event notification callbacks are expected from the BSP for the same operation.

The following table summarizes the compatibility between types of GUI events, moment values, and responses.

Table 4 — GUI responses and GUI events

Response from the application	GUI select event (before start)	GUI state event (before start)	GUI progress event (before start)	GUI progress event (during)	GUI progress event (after end)	GUI state event (after end)	GUI select event (after end)
BioAPI_GUI_RESPONSE_DEFAULT	X	X	X	X	X	X	X
BioAPI_GUI_RESPONSE_CYCLE_START	X						
BioAPI_GUI_RESPONSE_SUBOP_START		X					
BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE			X	X	X		
BioAPI_GUI_RESPONSE_SUBOP_NEXT						X	
BioAPI_GUI_RESPONSE_OP_COMPLETE							X
BioAPI_GUI_RESPONSE_PROGRESS_ABORT			X	X	X		
BioAPI_GUI_RESPONSE_RECAPTURE						X	
BioAPI_GUI_RESPONSE_CYCLE_RESTART		X	X	X	X	X	X
BioAPI_GUI_RESPONSE_OP_CANCEL	X	X	X	X	X	X	X

7.53 BioAPI_GUI_STATE (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

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 7.54.

7.54 BioAPI_GUI_STATE_CALLBACK (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

7.54.1 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.

7.54.2 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.

7.54.3 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.8 on User Interface Considerations.

7.55 BioAPI_GUI_STREAMING_CALLBACK (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

7.55.1 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.

7.55.2 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.

7.55.3 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.8 on User Interface Considerations.

7.56 BioAPI_GUI_SUBOPERATION (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

An enumeration of the possible types of suboperations performed by a BSP during an operation, to be reported to the application in GUI event notifications.

```
typedef uint8_t BioAPI_GUI_SUBOPERATION;

   #define BioAPI_GUI_SUBOPERATION_CAPTURE          (1)
   #define BioAPI_GUI_SUBOPERATION_PROCESS          (2)
   #define BioAPI_GUI_SUBOPERATION_CREATETEMPLATE   (3)
   #define BioAPI_GUI_SUBOPERATION_VERIFYMATCH      (4)
   #define BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH    (5)
```

The values of this type occur in GUI event notifications and identify a type of suboperation of the current operation for which the GUI event has been generated.

The following table shows the types of suboperations (and their number) performed by a BSP during each type of operation (in order). For the operations BioAPI_GUI_OPERATION_CAPTURE, BioAPI_GUI_OPERATION_VERIFY, BioAPI_GUI_OPERATION_IDENTIFY, and BioAPI_GUI_OPERATION_ENROLL, the number of suboperations is relative to a complete suboperation cycle (see 7.47) and does not take account of possible recaptures done within the cycle (see 7.52).

Table 5 — Suboperations performed in each operation

Operation	Suboperations
BioAPI_GUI_OPERATION_CAPTURE with the purpose of verification or identification	BioAPI_GUI_SUBOPERATION_CAPTURE (one) BioAPI_GUI_SUBOPERATION_PROCESS (zero or one)
BioAPI_GUI_OPERATION_CAPTURE with the purpose of enrollment (when the enroll type is Test-Verify)	BioAPI_GUI_SUBOPERATION_CAPTURE (two) BioAPI_GUI_SUBOPERATION_CREATETEMPLATE (one) BioAPI_GUI_SUBOPERATION_PROCESS (one) BioAPI_GUI_SUBOPERATION_VERIFYMATCH (one)
BioAPI_GUI_OPERATION_CAPTURE with the purpose of enrollment (when the enroll type is Multiple-Capture)	BioAPI_GUI_SUBOPERATION_CAPTURE (multiple) BioAPI_GUI_SUBOPERATION_CREATETEMPLATE (one)
BioAPI_GUI_OPERATION_PROCESS	BioAPI_GUI_SUBOPERATION_PROCESS (one)
BioAPI_GUI_OPERATION_CREATETEMPLATE	BioAPI_GUI_SUBOPERATION_CREATETEMPLATE (one)
BioAPI_GUI_OPERATION_VERIFYMATCH	BioAPI_GUI_SUBOPERATION_VERIFYMATCH (one)
BioAPI_GUI_OPERATION_IDENTIFYMATCH	BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH (one)
BioAPI_GUI_OPERATION_VERIFY	BioAPI_GUI_SUBOPERATION_CAPTURE (one) BioAPI_GUI_SUBOPERATION_PROCESS (one) BioAPI_GUI_SUBOPERATION_VERIFYMATCH (one)

Table 5 *(continued)*

Operation	Suboperations
BioAPI_GUI_OPERATION_IDENTIFY	BioAPI_GUI_SUBOPERATION_CAPTURE (one) BioAPI_GUI_SUBOPERATION_PROCESS (one) BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH (one)
BioAPI_GUI_OPERATION_ENROLL (when the enroll type is Test-Verify)	BioAPI_GUI_SUBOPERATION_CAPTURE (two) BioAPI_GUI_SUBOPERATION_CREATETEMPLATE (one) BioAPI_GUI_SUBOPERATION_PROCESS (one) BioAPI_GUI_SUBOPERATION_VERIFYMATCH (one)
BioAPI_GUI_OPERATION_ENROLL (when the enroll type is Multiple-Capture)	BioAPI_GUI_SUBOPERATION_CAPTURE (multiple) BioAPI_GUI_SUBOPERATION_CREATETEMPLATE (one)

7.57 BioAPI_HANDLE

A unique identifier, returned on ***BioAPI_BSPAttach*/*BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater), that identifies an attached BioAPI BSP session.

```
typedef uint32_t BioAPI_HANDLE;
```

When the BioAPI version number in use is 2.1 or greater, the value zero shall not be used as a valid BSP attach session handle value. All other 32-bit unsigned integer values are permitted.

```
#define BioAPI_INVALID_BSP_HANDLE (0)
```

This definition applies only when the BioAPI version number in use is 2.1 or greater.

7.58 BioAPI_HASH_ALG (BioAPI 2.2)

Identifies the hash algorithm supported by a BioAPI Unit, which is used in the generation of ACBio instance. This BioAPI type shall be the XML value notation of ASN.1 identifier (see ISO/IEC 8824-1) assigned to hash algorithms. Example to specify the SHA-1 hash algorithm, the char would be "1.3.14.3.2.26" which is the XML value notation for.

```
typedef char *BioAPI_HASH_ALG;

   #define BioAPI_HASH_ALG_NOT_SUPPORTED NULL
```

7.59 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.

7.60 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)
```

7.61 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)
```

7.62 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;
```

7.63 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.

7.64 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)
```

7.65 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;
```

7.66 BioAPI_KEY_INFO (BioAPI 2.2)

7.66.1 Union giving information of cryptographic key of a BioAPI Unit or a biometric application which is used to encrypt/decrypt biometric data or to generate/validate MAC of BIR.

```
typedef union bioapi_key_info {
   BioAPI_KEY_TRANSPORT KTInfo;
   BioAPI_ASN1_ENCODED KEKInfo;
} BioAPI_KEY_INFO;
```

7.66.2 Definitions

— *KTInfo* – Key information when key management technique of key transport is used.

— *KEKInfo* – Key information when key management technique of previously distributed symmetric key-encryption keys is used.

NOTE For details of key management, see RFC 3852.

7.67 BioAPI_KEY_TRANSPORT (BioAPI 2.2)

7.67.1 Structure giving information of cryptographic key of a BioAPI Unit or a biometric application, which is used to encrypt/decrypt biometric data or to generate/.validate MAC of BIR, when key management technique of key transport is used.

```
typedef struct bioapi_key_transport {
   BioAPI_ASN1_ENCODED IssuerAndSerialNumber;
   BioAPI_ASN1_ENCODED Certificate;
} BioAPI_KEY_TRANSPROT;
```

7.67.2 Definitions

— *IssuerAndSerialNumber* – ASN.1 encoded data of ASN.1 type `IssuerAndSerialNumber` which contains the information of issuer and serial number of the X.509 certificate for the public key.

— *Certificate* – ASN.1 encoded data of ASN.1 type `Certificate` which contains X.509 certificate of the public key.

NOTE For details of the definitions of the types IssuerAndSerialNumber and Certificate, see RFC 3852.

7.68 BioAPI_MAC_ALG (BioAPI 2.2)

Identifies the MAC algorithm supported by a BioAPI Unit. Example to specify the HMAC algorithm with SHA-1, the char would be “1.3.6.1.5.5.8.1.2” which is the XML value notation for.

```
typedef char *BioAPI_MAC_ALG;

   #define BioAPI_MAC_ALG_NOT_SUPPORTED NULL
```

7.69 BioAPI_MAC_INFO (BioAPI 2.2)

7.69.1 Structure giving information of the MAC algorithm and key(s) of a BioAPI Unit or a biometric application which is used to generate/validate MAC of BIR.

```
typedef struct bioapi_mac_info {
   BioAPI_MAC_ALG MACAlg;
   BioAPI_KEY_INFO *MACKeyInfo;
   uint32_t *NumberOfElementsMacKeyInfo;
} BioAPI_MAC_INFO;
```

7.69.2 Definitions

— *MACAlg* – MAC algorithm.

— *MACKeyInfo* – An array of key information for message authentication code.

— *NumberOfElementsMacKeyInfo* – The number of elements in the array of MacKeyInfo.

7.70 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_SUBSCRIBETOGUIEVENTS       (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)
      #define BioAPI_TRANSFORM                  (0x00800000)
      #define BioAPI_PROCESSUSINGAUXBIRS        (0x01000000)
      #define BioAPI_VERIFYMATCHUSINGAUXBIRS    (0x02000000)
      #define BioAPI_DECIDE                     (0x04000000)
      #define BioAPI_FUSE                       (0x08000000)
      #define BioAPI_SECURITY                   (0x10000000)
```

The definition BioAPI_SETGUICALLBACKS applies only when the BioAPI version number in use is 2.0. The definition BioAPI_SUBSCRIBETOGUIEVENTS applies only when the BioAPI version number in use is 2.1 or greater. The definition BioAPI_TRANSFORM applies only when the BioAPI version number in use is 2.1 or greater.

NOTE This type is used as an element of BioAPI_BSP_SCHEMA.

7.71 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. This definition applies only when the BioAPI version number in use is 2.0.

```
#define   BioAPI_GUI_PROGRESS_EVENTS      (0x00000020)
```
If set, the BSP provides GUI progress events. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#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 a payload during ***BioAPI_Enroll*** or ***BioAPI_CreateTemplate*** and returns the payload upon successful ***BioAPI_Verify, BioAPI_VerifyMatch***), or ***VerifyMatchUsingAuxBIRs*** (BioAPI 2.2 or greater)).

```
#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***, ***VerifyMatch***, or ***VerifyMatchUsingAuxBIRs*** (BioAPI 2.2 or greater) operation.

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

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

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

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

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

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

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

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

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

```
#define  BioAPI_IDENTIFYINDICATOR        (0x00200000)
```

If set, the BSP supports progress indication during identification. This definition applies only when the BioAPI version number in use is 2.1 or greater.

NOTE This type is used as an element of BioAPI_BSP_SCHEMA.

7.72 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)
```

7.73 BioAPI_QUALITY

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

```
typedef int8_t BioAPI_QUALITY;
```

7.73.2 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).

7.73.3 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.

7.73.4 There are two objectives in providing BioAPI_QUALITY feedback to the biometric application.

a) 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).

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

7.73.5 Quality scores in the range 0-100 have the following general interpretation.

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

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

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

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

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

7.74 BioAPI_RETURN

7.74.1 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)
```

7.74.2 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.)*

7.75 BioAPI_SECURITY_OPTIONS_MASK (BioAPI 2.2)

A mask that indicates what security options are supported by the BioAPI Unit.

```
typedef uint32_t BioAPI_SECURITY_OPTIONS_MASK;

#define BioAPI_ENCRYPTION (0x00000001)
```
If set, indicates that the BioAPI Unit supports encryption.

```
#define BioAPI_MAC (0x00000002)
```
If set, indicates that the BioAPI Unit supports MAC generation.

```
#define BioAPI_DIGITAL_SIGNATURE (0x00000004)
```
If set, indicates that the BioAPI Unit supports digital signature.

```
#define BioAPI_ACBio_GENERATION_WITH_MAC (0x00000010)
```
If set, indicates that the BioAPI Unit supports ACBio generation using MAC.

```
#define BioAPI_ACBio_GENERATION_WITH_DIGITAL_SIGNATURE (0x00000020)
```
If set, indicates that the BioAPI Unit supports ACBio generation using digital signature.

7.76 BioAPI_SECURITY_PROFILE (BioAPI 2.2)

7.76.1 Structure giving information of the cryptographic algorithms and keys of a BioAPI Unit or a biometric application which is used to encrypt/decrypt biometric data or to generate/validate the MAC or digital signature of the BIR and also giving the information of the hash algorithm, the information about the MAC generation, and the digital signature used in ACBio generation. When this structure is used in the structure of BioAPI_UNIT_SCHEMA (BioAPI 2.2) as the output parameter of ***BioAPI_QueryUnits***,

the parameters in this structure indicate the information supported in the BioAPI Unit. On the other hand, when this structure is used as the parameter of ***BioAPI_BSPAttachSecure***, the parameters in this structure indicates the information which is to be used in the execution of security operations.

```
typedef struct bioapi_security_profile {
   BioAPI_SECURITY_OPTIONS_MASK SupportedSecOption;
   BioAPI_ENCRYPTION_INFO **ENCInfo;
   uint32_t *NumberOfElementsEncInfo;
   BioAPI_MAC_INFO **MACInfo;
   uint32_t *NumberOfElementsMacInfo;
   BioAPI_DIGITAL_SIGNATURE_ALG *SIGNAlg;
   uint32_t *NumberOfElementsSIGNAlg;
   BioAPI_OPERATIONS_MASK ACBioOption;
   BioAPI_HASH_ALG **HASHAlgForACBio;
   uint32_t *NumberOfElementsHASHAlgForACBio;
   BioAPI_MAC_INFO **MACInfoForACBio;
   uint32_t *NumberOfElementsMACInfoForACBio;
   BioAPI_DIGITAL_SIGNATURE_ALG *SIGNAlgForACBio;
   uint32_t *NumberOfElementsSIGNAlgForACBio;
} BioAPI_SECURITY_PROFILE;
```

7.76.2 Definitions

— *SupportedSecOption* – A mask which indicates which security options are supported or to be executed by the BSP Unit.

— *ENCInfo* – Encryption information used in the encryption of the BDB.

— *NumberOfElementsEncInfo* – The number of elements in the array of ENCInfo.

— *MACInfo* – MAC information used to keep the integrity of the BIR.

— *NumberOfElementsMacInfo* – The number of elements in the array of MACInfo.

— *SIGNAlg* – Digital signature algorithm used to keep the integrity of the BIR.

— *NumberOfElementsSIGNAlg* – The number of elements in the array of SIGNAlg.

— *ACBioOption* – A mask which indicates which security options of MAC or digital signature are supported or to be executed by the BSP Unit.

— *HASHAlgForACBio* – Hash algorithm used to generate ACBio instances.

— *NumberOfElementsHASHAlgForACBio* – The number of elements in the array of HASHAlgForACBio.

— *MACInfoForACBio* – MAC information used to generate ACBio instances.

— *NumberOfElementsMACInfoForACBio* – The number of elements in the array of MACInfoForACBio.

— *SIGNAlgForACBio* – Digital signature algorithm used to generate ACBio instances.

— *NumberOfElementsSIGNAlgForACBio* – The number of elements in the array of SIGNAlgForACBio.

7.77 BioAPI_DIGITAL_SIGNATURE_ALG (BioAPI 2.2)

Identifies the digital signature algorithm supported by a BioAPI Unit. This BioAPI type shall be the XML value notation of ASN.1 identifier (see ISO/IEC 8824-1) assigned to digital signature algorithms. Example to specify the digital signature algorithm using SHA1 with RSA according to ISO/IEC 9796-2, the char would be “1.3.36.3.4.3.2.1” which is the XML value notation for.

```
typedef char *BioAPI_DIGITAL_SIGNATURE_ALG;

   #define BioAPI_DIGITAL_SIGNATURE_ALG_NOT_SUPPORTED NULL
```

7.78 BioAPI_STRING

7.78.1 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];
```

7.78.2 This string shall consist of ISO/IEC 10646 characters encoded in UTF-8 (see ISO/IEC 10646:2017, Annex D).

7.79 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].

7.80 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)
```

7.81 BioAPI_UNIT_LIST_ELEMENT

7.81.1 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;
```

7.81.2 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.

7.82 BioAPI_UNIT_SCHEMA

7.82.1 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;
    BioAPI_EVENT_MASK 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.

7.82.2 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 cryptographically authenticated by the BSP (or BFP).

— *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.

7.83 BioAPI_UNIT_SCHEMA (BioAPI 2.2)

7.83.1 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;
        BioAPI_EVENT_MASK 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_SECURITY_PROFILE *SecurityProfile;
   } 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.

7.83.2 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.

— *SecurityProfile* – Security profile of the BioAPI Unit.

7.84 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.

7.85 BioAPI_VERSION

This type is used to represent the version of the BioAPI or FPI specification to which components or data have been implemented. This type is used in the function BioAPI_Init, within the BIR header, and within schemas in the component registry.

The two following values are specified in this edition of this International Standard:

1) 32 decimal (20 hex), corresponding to a Major value of 2 and a Minor value of 0, and representing BioAPI 2.0;

2) 33 decimal (21 hex), corresponding to a Major value of 2 and a Minor value of 1, and representing BioAPI 2.1;

3) 34 decimal (22 hex), corresponding to a Major value of 2 and a Minor value of 2, and representing BioAPI 2.2.

```
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.

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.

7.86 GUI Events

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

7.86.1 BioAPI_GUI_SELECT_EVENT_HANDLER (BioAPI 2.1)

7.86.1.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioAPI_GUI_SELECT_EVENT_HANDLER)
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    BioAPI_GUI_ENROLL_TYPE     EnrollType,
    const void *GUISelectEventHandlerCtx,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
    int32_t MaxNumEnrollSamples,
    BioAPI_BIR_SUBTYPE_MASK SelectableInstances,
    BioAPI_BIR_SUBTYPE_MASK *SelectedInstances,
    BioAPI_BIR_SUBTYPE_MASK CapturedInstances,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response);
```

7.86.1.2 Description

This is a function pointer type for an application's GUI event handler function that is to handle GUI select event notification callbacks coming from a BSP through the framework. In order to receive GUI select event notifications, an application shall register a callback function of type BioAPI_GUI_SELECT_EVENT_HANDLER by providing the callback address of the function, along with a context address, in a call to ***BioAPI_SubscribeToGUIEvents*** (see 8.3.8).

The framework makes a callback to an application function of this type each time it receives an incoming callback to a function of type BioSPI_GUI_SELECT_EVENT_HANDLER which it exposes to a BSP. The parameters of the callback (except GUISelectEventHandlerCtx) shall be set from the parameters of the incoming callback with the same names; the parameter GUISelectEventHandlerCtx shall be set from the GUI select context address originally provided by the application in its call to ***BioAPI_SubscribeToGUIEvents***; and the callback address shall be set from the GUI select callback address originally provided by the application in the call to ***BioAPI_SubscribeToGUIEvents***.

A BSP can generate GUI select events only during the execution of a call to ***BioAPI_Capture***, ***BioAPI_Verify***, ***BioAPI_Identify***, or ***BioAPI_Enroll***. A GUI select event with the moment value BioAPI_GUI_MOMENT_BEFORE_START is generated before the start of each suboperation cycle, and a GUI select event with the moment value BioAPI_GUI_MOMENT_AFTER_END is generated after the end of each suboperation cycle (see 7.47).

The main purpose of the GUI select event notification generated before the start of a suboperation cycle is to inform the application that a new suboperation cycle is ready to start and to allow the application to select the instance(s) to be captured (where such selection is supported by the BSP and is consistent with the biometric modality in use). The main purpose of the GUI select event notification generated after the end of a suboperation cycle is to inform the application about the outcome of the suboperation cycle. The application shall respond to each GUI select notification by indicating whether the suboperation cycle should start, the operation should be considered completed, a new suboperation cycle should be started, or the operation should be canceled (see 7.52).

If the application specifies one or more instances in the *Subtype* parameter of the original function call (***BioAPI_Capture***, ***BioAPI_Enroll***, ***BioAPI_Verify***, or ***BioAPI_Identify***) and this parameter is supported by the BSP, then the parameter *SelectableInstances* is set based on the value of the *Subtype* parameter.

The GUI select event handler function and any functions invoked (directly or indirectly) by that function shall not call any BioAPI function.

If the application returns a value different from BioAPI_OK, the original function will immediately return to the caller with an error code.

7.86.1.3 Parameters

— *BSPUuid (input)* – A UUID identifying the BSP to which the GUI event is related.

— *UnitID (input)* – The unit ID of the sensor unit of the BSP to which the GUI event is related.

— *BSPHandle (input)* – The handle of the BSP attach session to which the GUI event is related.

— *EnrollType (input)* – The enroll type of the BSP. This parameter is only meaningful if Operation is BioAPI_GUI_OPERATION_ENROLL.

— *GUISelectEventHandlerCtx (input)* – The context address orginally provided by the subscriber application as part of the GUI event subscription.

— *Operation (input)* – A value indicating the type of operation for which the GUI event has been generated. Only the following values are permitted:

```
BioAPI_GUI_OPERATION_CAPTURE
BioAPI_GUI_OPERATION_VERIFY
BioAPI_GUI_OPERATION_IDENTIFY
BioAPI_GUI_OPERATION_ENROLL
```

— *Moment (input)* – A value indicating whether the GUI event has been generated before the start of the suboperation cycle or after the end of the suboperation cycle. Only the following values are permitted:

```
BioAPI_GUI_MOMENT_BEFORE_START
BioAPI_GUI_MOMENT_AFTER_END
```

— *ResultCode (input)* – This parameter is significant only if Moment is BioAPI_GUI_MOMENT_AFTER_END, and shall be set to zero otherwise. When significant, it shall contain the result code of the suboperation cycle that has just ended. The result code can be either BioAPI_OK or a BioAPI error code.

— *MaxNumEnrollSamples (input)* – This parameter is significant only if Moment is BioAPI_GUI_MOMENT_BEFORE_START. When significant, it shall contain the maximum number of samples that the BSP will be able to capture in the suboperation cycle that is about to start.

— *SelectableInstances (input)* – This parameter is significant only if Moment is BioAPI_GUI_MOMENT_BEFORE_START. When significant, it shall contain a bitmask that specifies which instance(s) of the biometric modality can be selected by the application for capturing in the suboperation cycle that is about to start. Some BSPs support the selection of multiple instances of a modality because they are able to create, in a single capture suboperation, a single BIR containing data pertaining to multiple instances.

— *SelectedInstances* (output) – This output parameter is significant only if Moment is BioAPI_GUI_MOMENT_BEFORE_START. When significant, it shall contain a bitmask that specifies which instance(s) of the biometric modality have been selected by the application for capturing in the suboperation cycle that is about to start.

— *CapturedInstances (input)* – This parameter is significant only if Moment is BioAPI_GUI_MOMENT_AFTER_END, and shall be set to zero otherwise. When significant, it shall contain a bitmask that specifies which instance(s) have been successfully captured by the BSP within the suboperation cycle that has just ended.

— *Text (input)* – A textual message, consisting of a string of ISO/IEC 10646 characters encoded in UTF-8, which is intended for display to the subject or to an operator. The content of the message, as well as the language in which it is expressed, are not standardized.

— Response (output) – A value indicating the response from the application to the GUI select event notification (see 7.52).

NOTE See also C.8.

7.86.1.4 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.

7.86.1.5 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED
```

7.86.2 BioAPI_GUI_STATE_EVENT_HANDLER (BioAPI 2.1)

7.86.2.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioAPI_GUI_STATE_EVENT_HANDLER)
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    const void *GUIStateEventHandlerCtx,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_SUBOPERATION Suboperation,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
    int32_t EnrollSampleIndex,
    const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response,
    int32_t *EnrollSampleIndexToRecapture);
```

7.86.2.2 Description

This is a function pointer type for an application's GUI event handler function that is to handle GUI state event notification callbacks coming from a BSP through the framework. In order to receive GUI state event notifications, an application shall register a callback function of type BioAPI_GUI_STATE_EVENT_HANDLER by providing the callback address of the function, along with a context address, in a call to ***BioAPI_SubscribeToGUIEvents*** (see 8.3.8).

The framework makes a callback to an application function of this type each time it receives an incoming callback to a function of type BioSPI_GUI_STATE_EVENT_HANDLER which it exposes to a BSP. The parameters of the callback (except *GUIStateEventHandlerCtx*) shall be set from the parameters of the incoming callback with the same names; the parameter *GUIStateEventHandlerCtx* shall be set from the GUI state context address originally provided by the application in its call to ***BioAPI_SubscribeToGUIEvents***; and the callback address shall be set from the GUI state callback address originally provided by the application in the call to ***BioAPI_SubscribeToGUIEvents***.

A BSP can generate GUI state events only during the execution of a call to ***BioAPI_Capture***, ***BioAPI_Verify***, ***BioAPI_Identify***, or ***BioAPI_Enroll***. A GUI state event with the moment value BioAPI_GUI_MOMENT_BEFORE_START is generated before the start of each suboperation, and a GUI state event with the moment value BioAPI_GUI_MOMENT_AFTER_END is generated after the end of the each suboperation (see 7.47).

The main purpose of the GUI state event notification generated before the start of a suboperation is to inform the application that a certain suboperation is ready to start. The main purpose of the GUI state event notification generated after the end of a suboperation is to inform the application about the outcome of the suboperation. The application shall respond to each GUI state event notification by indicating whether the suboperation should start, the suboperation cycle should proceed with the next suboperation, a new suboperation cycle should be started, or the entire operation should be canceled (see 7.52).

The GUI state event handler function and any functions invoked (directly or indirectly) by that function shall not call any BioAPI function.

If the application returns a value different from BioAPI_OK, the original function will immediately return to the caller with an error code.

7.86.2.3 Parameters

— *BSPUuid (input)* – A UUID identifying the BSP to which the GUI event is related.

— *UnitID (input)* – The unit ID of the sensor unit of the BSP to which the GUI event is related.

— *BSPHandle (input)* – The handle of the BSP attach session to which the GUI event is related.

— *GUIStateEventHandlerCtx (input)* – The context address originally provided by the subscriber application as part of the GUI event subscription.

— *Operation (input)* – A value indicating the type of operation for which the GUI event has been generated. Only the following values are permitted:

```
BioAPI_GUI_OPERATION_CAPTURE
BioAPI_GUI_OPERATION_VERIFY
BioAPI_GUI_OPERATION_IDENTIFY
BioAPI_GUI_OPERATION_ENROLL
```

— *Suboperation (input)* – A value indicating the type of suboperation for which the GUI event has been generated (the suboperation that is about to start or has just ended). The suboperation shall be one of those that are compatible with Operation (see Table 5).

— *Purpose (input)* – This parameter is significant only if Suboperation is BioAPI_GUI_SUBOPERATION_CAPTURE, and shall be set to zero otherwise. When significant, it shall indicate the purpose of the capture suboperation (any value of type BioAPI_BIR_PURPOSE).

— *Moment (input)* – A value indicating whether the GUI event has been generated before the start of the suboperation or after the end of the suboperation. Only the following values are permitted:

```
BioAPI_GUI_MOMENT_BEFORE_START
BioAPI_GUI_MOMENT_AFTER_END
```

— *ResultCode (input)* – This parameter is significant only if Moment is BioAPI_GUI_MOMENT_AFTER_END, and shall be set to zero otherwise. When significant, it shall contain the result code of the suboperation that has just ended. The result code can be either BioAPI_OK or a BioAPI error code.

— *EnrollSampleIndex (input)* – This parameter is significant only if Suboperation is BioAPI_GUI_SUBOPERATION_CAPTURE, Moment is BioAPI_GUI_MOMENT_BEFORE_START, and a sample for enrollment is about to be captured. Otherwise, it shall be set to zero. When significant, it shall contain the index (an integer from 1 onwards) of the sample to be captured for enrollment by the capture suboperation that is about to start.

— *Bitmaps (input)* – An array of bitmaps containing image(s) of the samples captured in the suboperation, which are intended for display to the subject or to an operator. This array usually contains zero or one bitmaps, but may contain multiple bitmaps if the BSP has the ability to capture multiple instances of a biometric modality in a single capture operation.

— *Text (input)* – A textual message, consisting of a string of ISO/IEC 10646 characters encoded in UTF-8, which is intended for display to the subject or to an operator. The content of the message, as well as the language in which it is expressed, are not standardized.

— *Response* (output) – A value indicating the response from the application to the GUI state event notification (see 7.52).

— *EnrollSampleIndexToRecapture* (output) – This parameter is significant only if Suboperation is BioAPI_GUI_SUBOPERATION_CREATETEMPLATE, Moment is BioAPI_GUI_MOMENT_AFTER_END, and Response is BioAPI_GUI_RESPONSE_RECAPTURE. When significant, it shall contain the index (a positive integer) of a sample previously captured for enrollment that the application requests to be recaptured.

EXAMPLE If a BSP provides the enroll type Multiple-Capture to the ***BioAPI_GUI_SELECT_EVENT_HANDLER*** function, it may capture three biometric samples in each enroll operation. The BSP generates GUI state events before the start and after the end of each capture suboperation (with *EnrollSampleIndex* set to 1, 2, and 3), followed by a GUI state event before the start and after the end of the create template suboperation. When the application receives the GUI state event notification before the start of the create template suboperation, it asks the user which sample is to be replaced. The user chooses the second sample, after reviewing the bitmaps on screen. The application sets *Response* to BioAPI_GUI_RESPONSE_RECAPTURE and *EnrollSampleIndexToRecapture* to 2, and returns from the notification callback. The BSP then generates another GUI state event before the start and after the end of a new capture suboperation, with *EnrollSampleIndex* set to 2, and finally another GUI state event before the start and after the end of a new create template suboperation.

NOTE See also C.8.

7.86.2.4 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.

7.86.2.5 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED
```

7.86.3 BioAPI_GUI_PROGRESS_EVENT_HANDLER (BioAPI 2.1)

7.86.3.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioAPI_GUI_PROGRESS_EVENT_HANDLER)
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    const void *GUIProgressEventHandlerCtx,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_SUBOPERATION Suboperation,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_GUI_MOMENT Moment,
    uint8_t SuboperationProgress,
    const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response);
```

7.86.3.2 Description

This is a function pointer type for an application's GUI event handler function that is to handle GUI progress event notification callbacks coming from a BSP through the framework. In order to receive GUI progress event notifications, an application shall register a callback function of type BioAPI_GUI_PROGRESS_EVENT_HANDLER by providing the callback address of the function, along with a context address, in a call to **BioAPI_SubscribeToGUIEvents** (see 8.3.8).

The framework makes a callback to an application function of this type each time it receives an incoming callback to a function of type BioSPI_GUI_PROGRESS_EVENT_HANDLER which it exposes to a BSP. The parameters of the callback (except *GUIProgressEventHandlerCtx*) shall be set from the parameters of the incoming callback with the same names; the parameter *GUIProgressEventHandlerCtx* shall be set from the GUI progress context address originally provided by the application in its call to ***BioAPI_SubscribeToGUIEvents***; and the callback address shall be set from the GUI progress callback address originally provided by the application in the call to ***BioAPI_SubscribeToGUIEvents***.

A BSP can generate GUI progress events only during the execution of a call to ***BioAPI_Capture***, ***BioAPI_Process***, ***BioAPI_CreateTemplate***, ***BioAPI_VerifyMatch***, ***BioAPI_IdentifyMatch***, ***BioAPI_Verify***, ***BioAPI_Identify***, or ***BioAPI_Enroll***. They can be generated at any time, even multiple times, during any suboperation.

The main purpose of the GUI progress event notification is to inform the application about the progress of a lengthy suboperation (such as a capture or an identify match suboperation) as percent of completion, and to send to the application streaming data (as a series of bitmaps) collected by the sensor. One possible use of this information is for the application to show the bitmaps and a progress bar to the subject or an operator. The application shall respond to each GUI progress notification by indicating whether the suboperation should continue or abort (see 7.52).

The GUI progress event handler function and any functions invoked (directly or indirectly) by that function shall not call any BioAPI function.

If the application returns a value different from BioAPI_OK, the BSP shall cancel the current suboperation and produce an error code. The next GUI event generated by the BSP shall be a GUI state event indicating that the suboperation has ended with that error code.

7.86.3.3 Parameters

— *BSPUuid (input)* – A UUID identifying the BSP to which the GUI event is related.

— *UnitID (input)* – The unit ID of the sensor unit of the BSP to which the GUI event is related.

— *BSPHandle (input)* – The handle of the BSP attach session to which the GUI event is related.

— *GUIProgressEventHandlerCtx (input)* – The context address originally provided by the subscriber application as part of the GUI event subscription.

— *Operation (input)* – A value indicating the type of operation for which the GUI event has been generated.

— *Suboperation (input)* – A value indicating the type of suboperation for which the GUI event has been generated. The suboperation shall be one of those that are compatible with Operation (see Table 5).

— *Purpose (input)* – This parameter is significant only if Suboperation is BioAPI_GUI_SUBOPERATION_CAPTURE, and shall be set to zero otherwise. When significant, it indicates the purpose of the suboperation (any value of type BioAPI_BIR_PURPOSE).

— *Moment (input)* – A value indicating whether the GUI event has been generated before the start of the suboperation, during the suboperation, or after the end of the suboperation.

— *SuboperationProgress (input)* – The percent of completion of the suboperation.

— *Bitmaps (input)* – An array of bitmaps containing image(s) of the samples captured in the suboperation, which are intended for display to the subject or to an operator. This array usually contains zero or one bitmaps, but may contain multiple bitmaps if the BSP has the ability to capture multiple instances of a biometric modality in a single capture operation.

— *Text (input)* – A textual message, consisting of a string of ISO/IEC 10646 characters encoded in UTF-8, which is intended for display to the subject or to an operator. The content of the message, as well as the language in which it is expressed, are not standardized.

— *Response* (output) – A value indicating the response from the application to the GUI select event notification (see 7.52).

NOTE See also C.8.

7.86.3.4 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.

7.86.3.5 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED
```

8 BioAPI functions

NOTE This clause is not applicable for framework-free systems, as BSPs provide, and applications interface directly with, the BioSPI interfaces specified in Clause 9.

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*** or to ***BioAPI_InitEndpoint*** (in BioAPI 2.1 or greater) 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*** or ***BioAPI_InitEndpoint*** call) or BioAPIERR_INCOMPATIBLE_VERSION, but will not reinitialize. A count of the number of successful ***BioAPI_Init*** or ***BioAPI_InitEndpoint*** 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*** or ***BioAPI_InitEndpoint*** (in BioAPI 2.1 or greater) 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*** or ***BioAPI_InitEndpoint*** (i.e., calls 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 anomally 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.

BioAPI_GetFrameworkInfo can be called before ***BioAPI_Init***.

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;

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*** or ***BioAPI_InitEndpoint*** (in BioAPI 2.1 or greater) for which a corresponding call to ***BioAPI_Terminate*** has not yet been made.

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

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

8.1.4.2 Parameters

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

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

8.1.4.3 Return Value

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

8.1.4.4 Errors

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

8.1.5 BioAPI_BSPLoad

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

8.1.5.1 Description

This function initializes a BSP using the ***BioSPI_BSPLoad*** (see clause 9.3.1.1). Initialization includes registering the application's event handler for the specified BSP and enabling all events except those that were disabled by the most recent call to ***BioAPI_EnableEventNotifications*** (in BioAPI 2.1 or greater). The application can choose to provide an event handler function to receive notification of events. Many applications can independently and concurrently load the same BSP, and each application can establish its own event handler. They will all receive notification of an event. The same or different event handlers can be used if an application loads multiple BSPs.

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

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

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

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

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

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

There is a "use count" on the establishment of event handlers; they have to be de-established (by ***BioAPI_BSPUnload***) as many times as they were established. When the BioAPI Framework calls BioSPI_BSPLoad, it receives from the BSP an "insert" event notification for each available BioAPI unit (of each category). If the biometric application has provided an event handler in the call to BioAPI_BSPLoad and has not disabled "insert" event notifications, the Framework will call back, in turn, the application's event handler. When the BioAPI version number in use is 2.1 or greater, insert notifications can be disabled by calling the function BioAPI_EnableEventNotifications (see also the NOTE in 7.37). This will indicate to the biometric application that it can go ahead and do a ***BioAPI_BSPAttach*/*BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater). If the hardware component for a specific functionality is not present, the "insert" event cannot be raised until the hardware component has been plugged in.

This function shall only be called if there is at least one call to ***BioAPI_Init*** or ***BioAPI_InitEndpoint*** (in BioAPI 2.1 or greater) for which a corresponding call to ***BioAPI_Terminate*** has not yet been made. The function ***BioAPI_BSPAttach*/*BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) can be invoked multiple times for each call to ***BioAPI_BSPLoad***.

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

8.1.5.2 Parameters

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

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

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

8.1.5.3 Return Value

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

8.1.5.4 Errors

```
BioAPIERR_BSP_LOAD_FAIL
BioAPIERR_INVALID_UUID
```

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

8.1.6 BioAPI_BSPUnload

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

8.1.6.1 Description

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

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

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

This function should not be called by the application if there is a call to ***BioAPI_BSPAttach*/*BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) for which a corresponding call to ***BioAPI_BSPDetach*** (for a given BSP handle) has not yet been made. However, should this function be called while the BSP is still attached, then for each call to ***BioAPI_BSPAttach*/*BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) without a corresponding call to ***BioAPI_BSPDetach***, the actions relative to the missing corresponding call to ***BioAPI_BSPDetach*** shall be implicitly performed by the BioAPI Framework (as though the corresponding function had been called at that time), followed by the actions relative to ***BioAPI_BSPUnload,*** (i.e., the Framework will detach the BSP prior to unloading it).

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

8.1.6.2 Parameters

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

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

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

8.1.6.3 Return Value

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

8.1.6.4 Errors

```
BioAPIERR_INVALID_UUID
BioAPIERR_BSP_NOT_LOADED
```

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

8.1.7 BioAPI_BSPAttach

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

8.1.7.1 Description

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

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

When the BioAPI version number in use is 2.1 or greater, if a BioAPI unit used in an attach session becomes unavailable, the attach session shall be invalidated. When an attach session has been invalidated, the only functions that may be legally called specifying the handle of that attach session are ***BioAPI_BSPDetach*** and ***BioAPI_GetBIRFromHandle***. All other functions shall return BioAPIERR_INVALID_ATTACH_SESSION.

If a BSP supports events, it may generate a "remove" event when a BioAPI unit becomes unavailable. When the BioAPI version number in use is 2.1 or greater, if the application has set up an event handler and has not disabled "remove" event notifications, it will receive the "remove" event notification, and will be able to infer that all the attach sessions (if any) that were using the removed BioAPI unit are now invalid. The application can then either terminate those attach sessions by calling ***BioAPI_BSPDetach***, or be prepared to receive an BioAPIERR_INVALID_ATTACH_SESSION error from a subsequent function call, and then call ***BioAPI_BSPDetach***.

NOTE If a BioAPI unit was implicitly selected into an attach session (either through the use of BioAPI_DONT_CARE or by not including the category of that unit in the list provided as input to ***BioAPI_BSPAttach*/*BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater), the application may not know which attach sessions were using the BioAPI unit that has been removed, and thus may not be able to terminate those attach sessions without first getting BioAPIERR_INVALID_ATTACH_SESSION errors from subsequent function calls.

8.1.7.2 Parameters

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

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

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

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

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

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

NOTE 1 Any subsequent calls requiring use of a BioAPI Unit of this category will fail with an error return.

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

NOTE 2 Only one BioAPI Unit of each category can be attached by a biometric application for each BSP attach session at any time.

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

8.1.7.3 Return Value

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

8.1.7.4 Errors

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

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

8.1.8 BioAPI_BSPAttachSecure (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioAPI_BSPAttachSecure
   (const BioAPI_UUID *BSPUuid,
   BioAPI_VERSION Version,
   const BioAPI_ACBio_PARAMETERS *ACBioParameters,
   const BioAPI_UNIT_LIST_ELEMENT *UnitList,
   const BioAPI_SECURITY_PROFILE *SecurityProfileList,
   uint32_t NumUnits,
   BioAPI_HANDLE *NewBSPHandle);
```

8.1.8.1 Description

This function initiates a BSP attach secure session and verifies that the version of the BSP expected by the application is compatible with the version on the system. The caller shall specify a list of zero

or more BioAPI Units that the BSP is to use in the attach session being created, and also specify the security profile information to be used in biometric operations afterwards.

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

The challenge given in ACBioParameters shall be updated every time of the execution of biometric verification. Therefore another call of BioAPI_BSPAttachSecure is necessary after a call of BioAPI_VerifyMatch, BioAPI_Decide, or BioAPI_Verify.

8.1.8.2 Parameters

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

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

— *ACBioParameters (input*/optional*)* – Parameters about ACBio generation. A NULL pointer shall be set if no generation of ACBio instances is necessary. The BSP shall manage the value of BPUIOIndexOutput whose initial value is given as *InitialBPUIOIndexOutput, a member of this parameter. When a biometric operation function, BioAPI_Capture for example, is called for the first time, the value *InitialBPUIOIndexOutput shall be used as BPUIOIndexOutput. When the function is called again before another call of BioAPI_BSPAttachSecure, the value of BPUIOIndexOutput shall be incremented by 1 and used. On the other hand, BPUIOIndexInput is given in the BIR to which the pointer is given as a parameter of the function call, if there is. It is the value of the field bpuIOIndex of subBlockForACBio in the Security Block of the BIR.

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

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

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

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

NOTE 1 Any subsequent calls requiring use of a BioAPI Unit of this category will fail with an error return.

— *SecurityProfileList (input)* – a pointer to a buffer containing a list of BioAPI_SECURITY_PROFILE structures which contain security information to be used in security operations by BioAPI Units. The order of this list shall keep that of the *UnitList.*

— *NumUnits (input)* – The number of BioAPI Unit elements in the list that the pointer *UnitList* is pointing to.

NOTE 2 Only one BioAPI Unit of each category can be attached by a biometric application for each BSP attach secure session at any time.

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

8.1.8.3 Return Value

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

8.1.8.4 Errors

```
BioAPIERR_INCOMPATIBLE_VERSION
BioAPIERR_BSP_NOT_LOADED
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_INVALID_UUID
BioAPIERR_UNIT_IN_USE
BioAPIERR_INVALID_CATEGORY
BioAPIERR_SECURITY_CHALLENGE_NOT_SET
BioAPIERR_SECURITY_BPUIOINDEX_NOT_SET
BioAPIERR_SECURITY_SUPREMUM_BPUIOINDEX_NOT_SET
BioAPIERR_SECURITY_PROFILE_NOT_SET
BioAPIERR_SECURITY_ENCRYPTION_ALG_NOT_SUPPORTED
BioAPIERR_SECURITY_ENCRYPTION_ALG_NOT_SET
BioAPIERR_SECURITY_ENCKEY_NOT_SET
BioAPIERR_SECURITY_MAC_ALG_NOT_SUPPORTED
BioAPIERR_SECURITY_MACKEY_NOT_SET
BioAPIERR_SECURITY_MAC_ALG_NOT_SET
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_ALG_NOT_SUPPORTED
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_ALG_NOT_SET
BioAPIERR_SECURITY_HASH_ALG_ACBIO_NOT_SUPPORTED
BioAPIERR_SECURITY_HASH_ALG_ACBIO_NOT_SET
BioAPIERR_SECURITY_MAC_ALG_ACBIO_NOT_SUPPORTED
BioAPIERR_SECURITY_MACKEY_ACBIO_NOT_SET
BioAPIERR_SECURITY_MAC_ALG_ACBIO_NOT_SET
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_ALG_ACBIO_NOT_SUPPORTED
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_ALG_ACBIO_NOT_SET
```

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

8.1.9 BioAPI_BSPDetach

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

8.1.9.1 Description

This function detaches the biometric application from the BSP invocation.

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

This function shall only be called after ***BioAPI_BSPAttach*/*BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, and shall not be called more than once for the same BSP handle created by the call to ***BioAPI_BSPAttach*/*BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

8.1.9.2 Parameters

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

8.1.9.3 Return Value

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

8.1.9.4 Errors

`BioAPIERR_INVALID_BSP_HANDLE`

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

8.1.10 BioAPI_QueryUnits

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

8.1.10.1 Description

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

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

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

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

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

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

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

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

8.1.10.2 Parameters

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

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

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

NOTE The structure of BioAPI_UNIT_SCHEMA is dependent on the version of BioAPI. BioAPI_UNIT_SCHEMA of BioAPI 2.2 contains security information while that of BioAPI 2.1 or less does not.

8.1.10.3 Return Value

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

8.1.10.4 Errors

```
BioAPIERR_BSP_NOT_LOADED
BioAPIERR_INVALID_UUID
```

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

8.1.11 BioAPI_EnumBFPs

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

8.1.11.1 Description

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

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

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

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

This function shall only be called if there is at least one call to ***BioAPI_Init*** or ***BioAPI_InitEndpoint*** (in BioAPI 2.1 or greater) for which a corresponding call to ***BioAPI_Terminate*** has not yet been made.

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

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

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

8.1.11.2 Parameters

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

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

8.1.11.3 Return Value

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

8.1.11.4 Errors

See BioAPI Error Handling (Clause 11).

8.1.12 BioAPI_QueryBFPs

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

8.1.12.1 Description

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

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

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

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

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

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

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

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

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

8.1.12.2 Parameters

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

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

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

8.1.12.3 Return Value

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

8.1.12.4 Errors

`BioAPIERR_INVALID_BSP_HANDLE`

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

8.1.13 BioAPI_ControlUnit

BioAPI_RETURN BioAPI BioAPI_ControlUnit

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

8.1.13.1 Description

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

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

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

8.1.13.2 Parameters

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

— *UnitID (input) – ID of the BioAPI Unit.*

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

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

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

8.1.13.3 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.

8.1.13.4 Errors

```
BioAPIERR_BIOAPI_UNIT_NOT_INSERTED
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_UNIT_IN_USE
BioAPIERR_INVALID_BSP_HANDLE
```

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

8.1.14 BioAPI_Control (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

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

8.1.14.1 Description

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

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

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

8.1.14.2 Parameters

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

— *UnitID (input) – ID of the BioAPI Unit.*

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

— *InputData (input) – Address and length of a buffer containing the data to be sent to the BioAPI Unit related to the given ControlCode.*

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

8.1.14.3 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.

8.1.14.4 Errors

BioAPIERR_BIOAPI_UNIT_NOT_INSERTED

BioAPIERR_INVALID_UNIT_ID

BioAPIERR_UNIT_IN_USE

BioAPIERR_INVALID_BSP_HANDLE

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

8.1.15 BioAPI_Transform (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_Transform
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_UUID *OperationUUID,
    const BioAPI_INPUT_BIR *InputBIRs,
    uint32_t NumberOfInputBIRs,
    BioAPI_BIR_HANDLE **OutputBIRs,
    uint32_t *NumberOfOutputBIRs);
```

8.1.15.1 Description

This function transforms one or more BIRs provided as input into one or more output BIRs. The transformation to be performed is identified by the *OperationUUID* parameter.

This International Standard does not specify any standard values for the *OperationUUID* parameter and does not specify any particular transformations. It is assumed that *OperationUUID* values and their semantics will be specified either by the provider of a particular BSP or by additional specifications (such as an application profile).

The function performs the following actions (in order):

a) performs the transformation specified by the *OperationUUID* parameter using the BIR(s) provided as input and creating one or more output BIRs as required by the particular transformation;

b) allocates a memory block large enough to contain an array of elements of type BioAPI_BIR_HANDLE with as many elements as the number of output BIRs created in (a);

c) fills the array with the BIR handles of the output BIRs created in (a);

d) returns the address of the array in the *OutputBIR*s parameter and the size of the array in the NumberOfOutputBIRs parameter.

The memory block returned by the BioAPI function call shall be freed by the application using ***BioAPI_Free*** (see clause 8.7.2), and all the BIR handles present in the *OutputBIR*s array shall be destroyed using ***BioAPI_FreeBIRHandle***.

8.1.15.2 Parameters

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

— *OperationUUID (input)* – A UUID that identifies the transformation to be performed by the BSP.

— *InputBIRs (input)* - An array of BIRs (containing one or more BIRs) provided as input to the transformation.

— *NumberOfInputBIRs (input)* - The number of BIRs in the array InputBIRs.

— *OutputBIRs (output)* - A pointer to the address of an array of elements of type BioAPI_BIR_HANDLE containing the handles of the output BIRs created by the transformation.

— *NumberOfOutputBIRs (output)* - A pointer to the number of elements of the array *OutputBIR*s.

8.1.15.3 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.

8.1.15.4 Errors

BioAPIERR_BIOAPI_UNIT_NOT_INSERTED

BioAPIERR_UNIT_IN_USE

BioAPIERR_TRANSFORMATION_NOT_SUPPORTED

BioAPIERR_INVALID_BSP_HANDLE

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

8.1.16 BioAPI_LinkToEndpoint (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_LinkToEndpoint
        (const uint8_t *SlaveEndpointIRI);
```

8.1.16.1 Description

This function does not perform any actions, and is provided for modified specifications provided by interworking standards, which may specify the use of BioAPI frameworks present on multiple computers from within an application running on the same or a different computer. In those standards, this function may be used to connect the application to a framework running on a different computer, thus enabling the application to access the BSPs managed by that framework.

Applications conforming to this document shall either not call this function, or (if they call it) they shall set the parameter *SlaveEndpointIRI* to NULL. The function is provided to support interworking standards.

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

8.1.16.2 Parameters

— *SlaveEndpointIRI* – This parameter shall be ignored by frameworks conforming to this document and shall be set to NULL by an application. It is provided to support interworking standards.

8.1.16.3 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.

8.1.16.4 Errors

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

8.1.17 BioAPI_UnlinkFromEndpoint (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_UnlinkFromEndpoint
        (const uint8_t *SlaveEndpointIRI);
```

8.1.17.1 Description

This function does not perform any actions, and is provided for modified specifications provided by interworking standards, which may specify the use of additional BioAPI frameworks present on multiple computers from within an application running on the same or a different computer. In those standards, this function may be used to terminate the connection between the application and a framework.

Applications shall either not call this function, or (if they call it) they shall set the parameter SlaveEndpointIRI to NULL.

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

8.1.17.2 Parameters

SlaveEndpointIRI – This parameter shall be ignored by frameworks conforming to this document and shall be set to NULL by an application. It is provided to support interworking standards..

8.1.17.3 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.

8.1.17.4 Errors

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

8.1.18 BioAPI_EnumFrameworks (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_EnumFrameworks
    (BioAPI_FRAMEWORK_SCHEMA **FwSchemaArray,
    unit32_t *NumberOfElements);
```

8.1.18.1 Description

This function is similar to ***BioAPI_GetFrameworkInfo***, but allows for information about multiple BioAPI frameworks to be returned when using a specification provided by an interworking standard.

For the specification provided by this document, this function always returns information about a single framework, and is therefore equivalent to ***BioAPI_GetFrameworkInfo*** (albeit with different parameters).

This function provides information about all the BioAPI frameworks that are currently visible to the application (only one unless an interworking standard is in use). It performs the following actions (in order):

a) allocates a memory block large enough to contain an array of elements of type BioAPI_FRAMEWORK_SCHEMA with as many elements as the number of visible frameworks;

b) fills the array with the framework schemas of all visible frameworks;

c) returns the address of the array in the *FwSchemaArray* 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*** or ***BioAPI_InitEndpoint*** 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 a BSP.

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

8.1.18.2 Parameters

— *FwSchemaArray (output)* – A pointer to the address of the array of elements of type BioAPI_FRAMEWORK_SCHEMA (allocated by the framework) containing the framework schema information.

— *NumberOfElements (output)* – A pointer to the number of elements of the array (which is the number of framework currently visible to the application - only one unless an interworking standard is in use).

8.1.18.3 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.

8.1.18.4 Errors

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

8.2 Data Handle Operations

8.2.1 BioAPI_FreeBIRHandle

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

8.2.1.1 Description

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

This function shall only be called after ***BioAPI_BSPAttach/BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, and shall not be called after ***BioAPI_BSPDetach*** has been called for the BSP handle created by the call to ***BioAPI_BSPAttach/BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

8.2.1.2 Parameters

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

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

8.2.1.3 Return Value

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

8.2.1.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BSP_HANDLE
```

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

8.2.2 BioAPI_GetBIRFromHandle

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

8.2.2.1 Description

This function returns the BIR associated with a BIR handle returned by a BSP. If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, the returned BIR shall contain the security block. The BIR handle is freed by the BSP before the function returns.

8.2.2.2 Parameters

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

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

— *BIR (output) – Pointer to the retrieved BIR.*

8.2.2.3 Return Value

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

8.2.2.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BSP_HANDLE
```

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

8.2.3 BioAPI_GetHeaderFromHandle

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

8.2.3.1 Description

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

8.2.3.2 Parameters

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

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

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

8.2.3.3 Return Value

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

8.2.3.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BSP_HANDLE
```

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

8.3 Callback and Event Operations

8.3.1 BioAPI_EnableEvents

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

8.3.1.1 Description

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

function is not recommended when the BioAPI version number in use is 2.1 or greater. The function ***BioAPI_EnableEventNotifications*** should be used in that case.

In some cases, INSERT events cannot be suppressed (see clause 7.37).

8.3.1.2 Parameters

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

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

8.3.1.3 Return Value

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

8.3.1.4 Errors

See BioAPI Error Handling (Clause 11).

8.3.2 BioAPI_SetGUICallbacks (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

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

8.3.2.1 Description

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

NOTE Not all BSPs support the provision of streaming data.

8.3.2.2 Parameters

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

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

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

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

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

NOTE 1 Function sub-types for BioAPI_GUI_STATE_CALLBACK and BioAPI_GUI_STREAMING_CALLBACK are defined in clauses 7.41 and 7.46 respectively.

NOTE 2 See also clause C.8 on User Interface Considerations.

8.3.2.3 Return Value

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

8.3.2.4 Errors

```
BioAPIERR_INVALID_POINTER
BioAPIERR_INVALID_BSP_HANDLE
```

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

8.3.3 BioAPI_NotifyGUIProgressEvent (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_NotifyGUIProgressEvent
    (const uint8_t *SubscriberEndpointIRI,
    const BioAPI_UUID *GUIEventSubscriptionUuid,
    const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_SUBOPERATION Suboperation,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_GUI_MOMENT Moment,
    uint8_t SuboperationProgress,
    const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response);
```

8.3.3.1 Description

This function enables an application to request that a GUI progress event, generated by the application, be directed to a specified named GUI event subscription. A GUI event passed to this function is called an application-generated GUI event.

When this function is called, the framework shall search all existing named GUI event subscriptions that were created by the application identified by *SubscriberEndpointIRI* (*NULL* meaning the current application), looking for a subscription where the callback address for the GUI progress event is not *NULL* and the GUI event subscription UUID is the same as *GUIEventSubscriptionUuid*. If there is a matching subscription, then the framework shall make a callback to the GUI progress event handler specified in that subscription, setting the input parameters of the callback from the input parameters of the ***BioAPI_NotifyGUIProgressEvent*** call with the same names. After returning from the callback, the framework shall copy the output parameters of the callback to the output parameters of the ***BioAPI_NotifyGUIProgressEvent*** call with the same names.

If no matching named GUI event subscription exists, then the function shall return BioAPI_NO_GUI_EVENT_HANDLER.

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

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

8.3.3.2 Parameters

SubscriberEndpointIRI – An IRI that identifies the application that created one or more named GUI event subscriptions. This shall be NULL if that application is the current application. An application can learn the subscriber endpoint IRIs of other applications that are using the framework (and have created one or more named GUI event subscriptions for a given BSP) by calling ***BioAPI_QueryGUIEventSubscriptions***. If there are no other applications using the framework, the subscriber endpoint IRI will be NULL in all named subscriptions returned by ***BioAPI_QueryGUIEventSubscriptions***.

GUIEventSubscriptionUuid – A UUID that identifies a named GUI event subscription. This shall not be NULL. In some cases, the application has gotten this UUID from the information returned by a prior call to ***BioAPI_QueryGUIEventSubscriptions***; in other cases, the application provides a known UUID.

The other parameters define the GUI progress event generated by the application.

8.3.3.3 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.

8.3.3.4 Errors

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

8.3.4 BioAPI_NotifyGUISelectEvent (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_NotifyGUISelectEvent
    (const uint8_t *SubscriberEndpointIRI,
    const BioAPI_UUID *GUIEventSubscriptionUuid,
    const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
    int32_t MaxNumEnrollSamples,
    BioAPI_BIR_SUBTYPE_MASK SelectableInstances,
    BioAPI_BIR_SUBTYPE_MASK *SelectedInstances,
    BioAPI_BIR_SUBTYPE_MASK CapturedInstances,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response);
```

8.3.4.1 Description

This function enables an application to request that a GUI select event, generated by the application, be directed to a specified named GUI event subscription. A GUI event passed to this function is called an application-generated GUI event.

When this function is called, the framework shall search all existing named GUI event subscriptions that were created by the application identified by *SubscriberEndpointIRI* (*NULL* meaning the current application), looking for a subscription where the callback address for the GUI select event is not *NULL* and the GUI event subscription UUID is the same as *GUIEventSubscriptionUuid*. If there is a matching subscription, then the framework shall make a callback to the GUI select event handler specified in that subscription, setting the input parameters of the callback from the input parameters of the ***BioAPI_NotifyGUISelectEvent*** call with the same names. After returning from the callback, the framework shall copy the output parameters of the callback to the output parameters of the ***BioAPI_NotifyGUISelectEvent*** call with the same names.

If no matching named GUI event subscription exists, then the function shall return BioAPI_NO_GUI_EVENT_HANDLER.

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

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

8.3.4.2 Parameters

SubscriberEndpointIRI (input, optional) – An IRI that identifies the application that created one or more named GUI event subscriptions. This shall be NULL if that application is the current application. An

application can learn the subscriber endpoint IRIs of other applications that are using the framework (and have created one or more named GUI event subscriptions for a given BSP) by calling ***BioAPI_QueryGUIEventSubscriptions***. If there are no other applications using the framework, the subscriber endpoint IRI will be NULL in all named subscriptions returned by ***BioAPI_QueryGUIEventSubscriptions***.

GUIEventSubscriptionUuid (input) – A UUID that identifies a named GUI event subscription. This shall not be NULL. In some cases, the application has gotten this UUID from the information returned by a prior call to ***BioAPI_QueryGUIEventSubscriptions***; in other cases, the application provides a known UUID.

The other parameters define the GUI select event generated by the application.

8.3.4.3 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.

8.3.4.4 Errors

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

8.3.5 BioAPI_NotifyGUIStateEvent (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_NotifyGUIStateEvent
 (const uint8_t *SubscriberEndpointIRI,
 const BioAPI_UUID *GUIEventSubscriptionUuid,
 const BioAPI_UUID *BSPUuid,
 BioAPI_UNIT_ID UnitID,
 BioAPI_GUI_OPERATION Operation,
 BioAPI_GUI_SUBOPERATION Suboperation,
 BioAPI_BIR_PURPOSE Purpose,
 BioAPI_GUI_MOMENT Moment,
 BioAPI_RETURN ResultCode,
 int32_t EnrollSampleIndex,
 const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
 const uint8_t *Text,
 BioAPI_GUI_RESPONSE *Response,
 int32_t *EnrollSampleIndexToRecapture);
```

8.3.5.1 Description

This function enables an application to request that a GUI state event, generated by the application, be directed to a specified named GUI event subscription. A GUI event passed to this function is called an application-generated GUI event.

When this function is called, the framework shall search all existing named GUI event subscriptions that were created by the application identified by *SubscriberEndpointIRI* (NULL meaning the current application), looking for a subscription where the callback address for the GUI state event is not NULL and the GUI event subscription UUID is the same as GUIEventSubscriptionUuid. If there is a matching subscription, then the framework shall make a callback to the GUI state event handler specified in that subscription, setting the input parameters of the callback from the input parameters of the ***BioAPI_NotifyGUIStateEvent*** call with the same names. After returning from the callback, the framework shall copy the output parameters of the callback to the output parameters of the ***BioAPI_NotifyGUIStateEvent*** call with the same names.

If no matching named GUI event subscription exists, then the function shall return BioAPI_NO_GUI_EVENT_HANDLER.

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

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

8.3.5.2 Parameters

— *SubscriberEndpointIRI (input, optional)* – An IRI that identifies the application that created one or more named GUI event subscriptions. This shall be NULL if that application is the current application. An application can learn the subscriber endpoint IRIs of other applications that are using the framework (and have created one or more named GUI event subscriptions for a given BSP) by calling ***BioAPI_QueryGUIEventSubscriptions***. If there are no other applications using the framework, the subscriber endpoint IRI will be NULL in all named subscriptions returned by ***BioAPI_QueryGUIEventSubscriptions***.

— *GUIEventSubscriptionUuid (input)* – A UUID that identifies a named GUI event subscription. This shall not be NULL. In some cases, the application has obtained this UUID from the information returned by a prior call to ***BioAPI_QueryGUIEventSubscriptions***; in other cases, the application provides a known UUID.

The other parameters define the GUI state event generated by the application.

8.3.5.3 Return Value

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

8.3.5.4 Errors

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

8.3.6 BioAPI_QueryGUIEventSubscriptions (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_QueryGUIEventSubscriptions
 (const BioAPI_UUID *BSPUuid,
 BioAPI_GUI_EVENT_SUBSCRIPTION **GUIEventSubscriptionList,
 uint32_t *NumberOfElements);
```

8.3.6.1 Description

This function provides information about all existing named GUI event subscriptions for a given BSP. A named GUI event subscription is one that was created by a call to ***BioAPI_SubscribeToGUIEvents*** specifying a not NULL GUI event subscription UUID. The framework calls the event handlers specified in a named subscription to notify BSP-generated GUI events that have been redirected to that named subscription (see 8.3.7), and to notify application-generated GUI events (see 8.3.3, 8.3.4, and 8.3.5) directed to that named subscription.

By calling this function, an application can learn the subscriber endpoint IRIs of other applications that are using the framework (where this is supported by the implementation architecture of the framework) and have created named GUI event subscriptions for a given BSP. If there are no other applications using the framework, the subscriber endpoint IRIs will be NULL in all the named subscriptions returned by this function.

An application can use the information returned by a call to this function (subscriber endpoint IRI, GUI event subscription UUID, and flags) in subsequent calls to ***BioAPI_RedirectGUIEvent*** (to request that certain subsequent BSP-generated GUI events be redirected to a specified named subscription), or in subsequent calls to ***BioAPI_NotifyGUISelectEvent***, ***BioAPI_NotifyGUIStateEvent***, etc. (to request that an application-generated GUI event be directed to a specified named subscription).

This function performs the following actions (in order):

a) allocates a memory block large enough to contain an array of elements of type BioAPI_GUI_EVENT_SUBSCRIPTION with as many elements as the number of existing named GUI event subscriptions for the given BSP;

b) fills the array with information about the named subscriptions;

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

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

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 8.7.2) when it is no longer needed by the application.

8.3.6.2 Parameters

— *GUIEventSubscriptionList (output)* – A pointer to the address of the array of elements of the type BioAPI_GUI_EVENT_SUBSCRIPTION (allocated by the framework) containing the named GUI event subscription information.

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

8.3.6.3 Return Value

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

8.3.6.4 Errors

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

8.3.7 BioAPI_RedirectGUIEvents (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_RedirectGUIEvents
   (const uint8_t *SubscriberEndpointIRI,
   const BioAPI_UUID *GUIEventSubscriptionUuid,
   BioAPI_HANDLE BSPHandle,
   BioAPI_BOOL GUISelectEventRedirected,
   BioAPI_BOOL GUIStateEventRedirected,
   BioAPI_BOOL GUIProgressEventRedirected);
```

8.3.7.1 Description

This function creates a "GUI event redirector" for the current application. GUI event redirectors are maintained by the framework at runtime (usually in an internal table), and enable an application to delegate the handling of BSP-generated GUI events either to another application that is using the framework (where this is supported by the implementation architecture of the framework), or to another component of the same application.

An application calling this function shall provide the subscriber endpoint IRI of the application that created the named GUI event subscription (NULL if it is the same application) and the GUI event subscription UUID of the subscription, as well as information specifying the scope of the GUI event redirector (what BSP attach session and which type(s) of GUI events are affected). All the parameter values provided in a call to this function shall become part of the GUI event redirector maintained by the framework.

Each call to this function establishes a new GUI event redirector, and does not modify or replace an existing redirector. GUI event redirectors specifying a given BSP attach session handle shall be destroyed automatically by the framework when the attach session is destroyed. At any time, the application can ask the framework to destroy an existing redirector by making a call to ***BioAPI_UnredirectGUIEvents*** providing the same parameters it provided in the corresponding call to ***BioAPI_RedirectGUIEvents***.

Multiple call to this function shall not result, at any time, in multiple GUI select event redirectors, multiple GUI state event redirectors, or multiple GUI progress event redirectors established for the same BSP handle. This is to ensure that, for any incoming GUI event, there will be no ambiguity as to how to redirect the event.

NOTE This implies that two redirectors with the same parameters are not allowed.

When the framework receives a call to this function that successfully creates a GUI event redirector, it shall call in turn the function ***BioSPI_SubscribeToGUIEvents*** of the BSP if and only if there were no GUI event subscriptions and no GUI event redirectors already established for that BSP before this call. Subsequent calls to ***BioAPI_RedirectGUIEvents*** will not result in further calls to ***BioSPI_SubscribeToGUIEvents*** unless all existing redirectors and subscriptions for that BSP have been destroyed.

When the framework receives a GUI select event notification callback from a BSP, it shall first search all existing GUI event redirectors, looking for a redirector where GUISelectEventRedirected has the value TRUE and the BSP handle is the same as the one in the GUI event. One of the two following subclauses applies.

If there is no matching redirector, then the framework shall search all anonymous GUI event subscriptions created by the application that created the BSP attach session, looking for a subscription with a not NULL callback address for the GUI select event and with either a BSP handle matching the BSP handle of the event or a BSP UUID matching the BSP UUID of the event. If there is a matching subscription, the framework shall call the GUI select event handler in that subscription, otherwise it shall return control to the original callback from the BSP with default output parameters.

If there is a matching redirector, then the framework shall search all existing named GUI event subscriptions that were created by the application identified by the subscriber endpoint IRI in the redirector (NULL meaning the current application), looking for a subscription with a not NULL callback address for the GUI select event, with a GUI event subscription UUID matching the GUI event subscription UUID in the redirector, and with a BSP UUID matching the BSP UUID of the event. If there is a matching subscription, then the framewo rk shall make a callback to the GUI select event handler specified in that subscription, otherwise it shall return control to the original callback from the BSP with default output parameters.

In both cases, after returning from the application callback, the framework shall copy the output parameters of the application callback to the output parameters of the framework callback with the same names.

The four subclauses above apply in the same way to GUI state and to GUI progress events.

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

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

8.3.7.2 Parameters

— *SubscriberEndpointIRI (input, optional)* – An IRI that identifies the application that created a named GUI event subscription. This shall be NULL if the subscriber application is the same as the current application.

— *GUIEventSubscriptionUuid (input)* – A UUID that identifies a named GUI event subscription. In some cases, an application has gotten this UUID from the information returned by a prior call to ***BioAPI_QueryGUIEventSubscriptions***. In other cases, the application provides a known UUID.

— *BSPHandle (input)* – The handle of a BSP attach session to which the redirector is limited. GUI events related to other BSP attach sessions will not be affected by this call.

— *GUISelectEventRedirected (input)* – Flag indicating whether GUI select events are in the scope of the redirector. GUI select events will be affected by this call if and only if this flag has the value TRUE.

— *GUIStateEventRedirected (input)* – Flag indicating whether GUI state events are in the scope of the redirector. GUI state events will be affected by this call if and only if this flag has the value TRUE.

— *GUIProgressEventRedirected (input)* – Flag indicating whether GUI progress events are in the scope of the redirector. GUI progress events will be affected by this call if and only if this flag has the value TRUE.

8.3.7.3 Return Value

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

8.3.7.4 Errors

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

8.3.8 BioAPI_SubscribeToGUIEvents (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_SubscribeToGUIEvents
   (const BioAPI_UUID *GUIEventSubscriptionUuid,
   const BioAPI_UUID *BSPUuid,
   const BioAPI_HANDLE *BSPHandle,
   BioAPI_GUI_SELECT_EVENT_HANDLER GUISelectEventHandler,
   const void *GUISelectEventHandlerCtx,
   BioAPI_GUI_STATE_EVENT_HANDLER GUIStateEventHandler,
   const void *GUIStateEventHandlerCtx,
   BioAPI_GUI_PROGRESS_EVENT_HANDLER GUIProgressEventHandler,
   const void *GUIProgressEventHandlerCtx);
```

8.3.8.1 Description

This function creates a "GUI event subscription" for the current application. GUI event subscriptions are maintained by the framework at runtime (usually as entries of an internal table), and enable an application to receive GUI event notifications at certain specified callback addresses.

The application shall provide the callback addresses of one to three GUI event handlers (a GUI select event handler, a GUI state event handler and a GUI progress event handler), as well as information specifying the scope of the subscription (either a loaded BSP or an attach session of a BSP), thus limiting the GUI event notifications that the framework will send to those GUI event handlers. All the parameter values provided in a call to this function shall become part of the GUI event subscription maintained by the framework.

In any call to this function, either a BSP UUID or a BSP handle (but not both) shall be provided. If a BSP handle is provided, the subscription is limited to GUI event notifications that carry that BSP handle. If a BSP UUID is provided, the subscription is limited to GUI event notifications that carry that BSP UUID, and may or may not also carry a BSP handle. (All GUI event notification that carry a BSP handle also carry the UUID of the BSP, but not vice versa.)

Each call to this function establishes a new GUI event subscription, and does not modify or replace an existing subscription. GUI event subscriptions specifying a BSP attach session handle shall be destroyed automatically by the framework when the attach session is destroyed. GUI event subscriptions specifying a BSP UUID shall be destroyed automatically by the framework when the BSP is unloaded. At any time, the application can ask the framework to destroy an existing subscription by making a call to ***BioAPI_UnsubscribeFromGUIEvents*** providing the same parameters that were provided in the corresponding call to ***BioAPI_SubscribeToGUIEvents***.

The context addresses provided in the call shall be passed back by the framework to the application in subsequent callbacks to the event handlers. This International Standard does not assign any meaning to the context addresses, but specifies them to satisfy the needs of some applications.

Multiple call to this function shall not result, at any time, in multiple GUI select event handlers, multiple GUI state event handlers, or multiple GUI progress event handlers, established for the same BSP handle or for the same BSP UUID. This includes the case in which one subscription specifies a BSP UUID and another subscription specifies the handle of an attach session of the same BSP. This is to ensure that, for any incoming GUI event, there will be no ambiguity as to which GUI event handler (if any) is to be called by the framework.

NOTE 1 This implies that two subscriptions with the same parameters are not allowed.

When the framework receives a call to this function that successfully creates a GUI event subscription for a given BSP, it shall call in turn the function ***BioSPI_SubscribeToGUIEvents*** of the BSP if and only if there were no GUI event subscriptions and no GUI event redirectors (see 8.3.7) already established for that BSP before this call. Subsequent calls to ***BioAPI_SubscribeToGUIEvents*** for the same BSP will not result in further calls to ***BioSPI_SubscribeToGUIEvents*** unless all existing subscriptions and redirectors for that BSP have been destroyed. A BSP does not need to know how many GUI event subscriptions the application is requesting for it, nor whether those subscriptions specify a BSP UUID or a BSP handle, nor the types, callback addresses, and context addresses of the GUI event handlers that are specified in those subscriptions.

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

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

NOTE 2 See also C.8.

8.3.8.2 Parameters

— *GUIEventSubscriptionUuid (input, optional)* – The identifier of the subscription. This parameter shall be NULL (the usual case) for an anonymous GUI event subscription (one concerning the reception of GUI events generated by a BSP and not redirected). This parameter shall be not NULL for a named subscription (one concerning the reception of redirected GUI event notifications and application-generated GUI event notifications).

— *BSPUuid (input, optional)* – A UUID identifying the BSP to which the subscription is limited.

— *BSPHandle (input, optional)* – The handle of a BSP attach session to which the subscription is limited. This shall be NULL if *GUIEventSubscriptionUuid* is not NULL (a named subscription cannot specify an attach session handle).

— *GUISelectEventHandler (input, optional)* – The callback address of an application function that is to receive GUI select event notifications from the framework.

— *GUISelectEventHandlerCtx (input)* – A context address that is to be passed back by the framework to the application on each callback to the GUI select event handler.

— *GUIStateEventHandler (input, optional)* – The callback address of an application function that is to receive GUI state event notifications from the framework.

— *GUIStateEventHandlerCtx (input)* – A context address that is to be passed back by the framework to the application on each callback to the GUI state event handler.

— *GUIProgressEventHandler (input, optional)* – The callback address of an application function that is to receive GUI progress event notifications from the framework.

— *GUIProgressEventHandlerCtx (input)* – A context address that is to be passed back by the framework to the application on each callback to the GUI progress event handler.

Exactly one of *BSPUuid* and *BSPHandle* shall be specified. At least one of *GUISelectEventHandler*, *GUIStateEventHandler*, and *GUIProgressEventHandler* shall be specified. A context address shall only be

specified if the corresponding callback address is specified. The parameters that are not specified shall be set to NULL.

8.3.8.3 Return Value

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

8.3.8.4 Errors

```
BioAPIERR_INVALID_POINTER
```

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

8.3.9 BioAPI_UnredirectGUIEvents (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_UnredirectGUIEvents
   (const uint8_t *SubscriberEndpointIRI,
   const BioAPI_UUID *GUIEventSubscriptionUuid,
   BioAPI_HANDLE BSPHandle,
   BioAPI_BOOL GUISelectEventRedirected,
   BioAPI_BOOL GUIStateEventRedirected,
   BioAPI_BOOL GUIProgressEventRedirected);
```

8.3.9.1 Description

This function destroys a GUI event redirector previously created by a call to ***BioAPI_RedirectGUIEvents***. The particular redirector that is destroyed is the one whose definition exactly matches the parameters of the call.

NOTE 1 A call to this function will not destroy a redirector, even if there is only one redirector, unless the parameters of the call have exactly the same values as those of the call to ***BioAPI_RedirectGUIEvents*** that created the redirector. The application needs to remember the value of those parameters.

When the framework receives a call to this function that successfully destroys a GUI event redirector, it shall call in turn the function ***BioSPI_UnsubscribeFromGUIEvents*** of the BSP if and only if there are no more GUI event subscriptions and no more GUI event redirectors established for that BSP.

Usually, applications will not need to call this function, because all GUI event redirectors for a given BSP attach session are automatically destroyed by the framework when the attach session is destroyed. This function can be used when the application needs to modify the current GUI event redirections but does not want to destroy an attach session.

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

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

NOTE 2 See also C.8.

8.3.9.2 Parameters

The parameters of this function have the same meaning as the parameters of the function ***BioAPI_RedirectGUIEvents*** with the same names, and are used by the framework to identify the existing GUI event redirector that is to be destroyed.

8.3.9.3 Return Value

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

8.3.9.4 Errors

`BioAPIERR_INVALID_POINTER`

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

8.3.10 BioAPI_UnsubscribeFromGUIEvents (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_UnsubscribeFromGUIEvents
 (const BioAPI_UUID *GUIEventSubscriptionUuid,
 const BioAPI_UUID *BSPUuid,
 const BioAPI_HANDLE *BSPHandle,
 BioAPI_GUI_SELECT_EVENT_HANDLER GUISelectEventHandler,
 const void *GUISelectEventHandlerCtx,
 BioAPI_GUI_STATE_EVENT_HANDLER GUIStateEventHandler,
 const void *GUIStateEventHandlerCtx,
 BioAPI_GUI_PROGRESS_EVENT_HANDLER GUIProgressEventHandler
 const void *GUIProgressEventHandlerCtx);
```

8.3.10.1 Description

This function destroys a GUI event subscription previously created by a call to ***BioAPI_SubscribeToGUIEvents***. The particular subscription that is destroyed is the one whose definition exactly matches the parameters of the call.

NOTE 1 A call to this function will not destroy a GUI event subscription, even if there is only one subscription, unless the parameters of the call have exactly the same values as those of the call to ***BioAPI_SubscribeToGUIEvents*** that created the subscription. The application needs to remember the value of those parameters.

When the framework receives a call to this function that successfully destroys a GUI event subscription, it shall call in turn the function ***BioSPI_UnsubscribeFromGUIEvents*** of the BSP if and only if there are no more GUI event subscriptions and no more GUI event redirectors (see 8.3.7) established for that BSP.

Usually, applications will not need to call this function, because all GUI event subscriptions for a given BSP UUID or BSP attach session handle are automatically destroyed by the framework when the attach session is destroyed or when the BSP is unloaded. This function can be used when the application needs to modify the current GUI event subscriptions but does not want to unload a BSP or destroy an attach session.

EXAMPLE This function can be used in the following cases: (1) an application uses the application-controlled GUI both at enrollment and at verification/identification, but with different GUI event handlers; (2) an application uses the BSP-controlled GUI at enrollment but the application-controlled GUI at verification/identification, or vice versa.

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

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

NOTE 2 See also C.8.

8.3.10.2 Parameters

The parameters of this function have the same meaning as the parameters of the function ***BioAPI_SubscribeToGUIEvents*** with the same names, and are used by the framework to identify the existing GUI event subscription that is to be destroyed.

8.3.10.3 Return Value

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

8.3.10.4 Errors

`BioAPIERR_INVALID_POINTER`

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

8.3.11 BioAPI_EnableEventNotifications (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_EnableEventNotifications
   (const BioAPI_UUID *BSPUuid,
   BioAPI_EVENT_MASK Events);
```

8.3.11.1 Description

This function enables the events specified by the Event Mask and coming from the BSP identified by the BSP UUID, and disables all other events coming from that BSP.

Events are enabled or disabled only for the application that calls this function. If there are other applications using the BioAPI Framework or the specified BSP at the same time, those applications will not be affected by the call to ***BioAPI_EnableEventNotifications***.

This function can be called at any time after ***BioAPI_Init*** or ***BioAPI_InitEndpoint***, even before the specified BSP has been loaded. The event mask established by a call to this function remains in effect until this function is called again for the same BSP.

This function shall only be called if there is at least one call to ***BioAPI_Init*** or ***BioAPI_InitEndpoint*** 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 a BSP.

8.3.11.2 Parameters

— *BSPUuid (input)* – The UUID of the biometric service provider.

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

8.3.11.3 Return Value

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

8.3.11.4 Errors

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

8.4 Biometric Operations

8.4.1 BioAPI_Capture

```
BioAPI_RETURN BioAPI BioAPI_Capture
   (BioAPI_HANDLE BSPHandle,
   BioAPI_BIR_PURPOSE Purpose,
   BioAPI_BIR_SUBTYPE Subtype,
   const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
   BioAPI_BIR_HANDLE *CapturedBIR,
```

```
int32_t Timeout,
BioAPI_BIR_HANDLE *AuditData);
```

8.4.1.1 Description

This function captures samples for the purpose specified, and the BSP returns either an "intermediate" type BIR (if the ***BioAPI_Process***, ***BioAPI_ProcessWithAuxBIR***, or ***BioAPI_ProcessUsingAuxBIRs*** (BioAPI 2.2 or greater) function needs to be called), or a "processed" BIR (if not). The *Purpose* is recorded in the header of the *CapturedBIR*. If *AuditData* is non-NULL, a BIR of type "raw" may be returned. The function returns handles to whatever data is collected, and all local operations can be completed through use of the handles. If the application needs to acquire the data either to store it in a BIR database or to send it to a server, the application can retrieve the data with the ***BioAPI_GetBIRFromHandle*** function.

By default, the BSP is responsible for providing the user interface associated with the capture operation. The application may, however, request control of the GUI "look-and-feel" by providing a GUI callback pointer in ***BioAPI_SetGUICallbacks***. See clause C.8 for additional explanation of user interface features.

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

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

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

8.4.1.2 Parameters

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

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

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

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

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

— *CapturedBIR (output)* – A handle to a BIR containing captured data. This data is either an "intermediate" type BIR (which can only be used by either the ***BioAPI_Process, BioAPI_CreateTemplate***, ***BioAPI_ProcessWithAuxBIR***, or ***BioAPI_ProcessUsingAuxBIRs*** (BioAPI 2.2 or greater) functions, depending on the purpose), or a "processed" BIR (which can be used directly by ***BioAPI_VerifyMatch*** or ***BioAPI_IdentifyMatch***, depending on the purpose).

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

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

8.4.1.3 Return Value

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

8.4.1.4 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_UNABLE_TO_CAPTURE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_TIMEOUT_EXPIRED
BioAPIERR_PURPOSE_NOT_SUPPORTED
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_UNIT_IN_USE
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
```

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

8.4.2 BioAPI_CreateTemplate

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

8.4.2.1 Description

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

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

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

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

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO

index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

8.4.2.2 Parameters

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

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

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

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

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

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

NOTE 1 Not all BSPs support storage of payloads.

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

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

8.4.2.3 Return Value

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

8.4.2.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BIR
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_UNABLE_TO_STORE_PAYLOAD
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_PURPOSE_NOT_SUPPORTED
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

8.4.3 BioAPI_Process

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

8.4.3.1 Description

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

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit or BFP which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

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

8.4.3.2 Parameters

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

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

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

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

8.4.3.3 Return Value

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

8.4.3.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BIR
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_PURPOSE_NOT_SUPPORTED
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_QUALITY_ERROR
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
```

```
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

8.4.4 BioAPI_ProcessWithAuxBIR (BioAPI2.0 and BioAPI2.1)

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

8.4.4.1 Description

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

NOTE 1 This capability may be used to support biometric match-on-card (MOC). See clause C.8 for a description of BioAPI use within the overall MOC process.

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

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

NOTE 2 This function is used only in BioAPI 2.1 or less. In BioAPI 2.2 or greater, ***BioAPI_ProcessUsingAuxBIRs*** (BioAPI 2.2) is used instead of this function.

8.4.4.2 Parameters

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

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

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

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

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

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

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

8.4.4.3 Return Value

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

8.4.4.4 Errors

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

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

8.4.5 BioAPI_ProcessUsingAuxBIRs (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioAPI_ProcessUsingAuxBIRs
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_INPUT_BIR *CapturedBIR,
    uint32_t NumberOfAuxBIRs,
    const BioAPI_INPUT_BIR *AuxBIRs,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
    BioAPI_BIR_HANDLE *ProcessedBIR);
```

8.4.5.1 Description

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

In addition to the intermediate BIR, this function takes as input a list of auxiliary BIRs. Each auxiliary BIR in the list may be an intermediate, processed, score, decision, or biographic BIR. The biometric service provider may modify the way it performs the processing operation based on the auxiliary BIRs provided as input and on the order in which they occur in the list. However, such modifications to the processing operation are not specified in this standard.

The main purposes of the auxiliary BIRs are:

a) to provide feedback to the biometric processing operation from prior processing, matching, fusion, or decision operations;

b) to modify the biometric processing operation based on statistical information (either specific to the subject, or specific to a population, or both).

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit or BFP which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

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

8.4.5.2 Parameters

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

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

— *NumberOfAuxBIRs (input)* – The number of elements (auxiliary BIRs) in the array pointed to by the parameter AuxBIRs.

— *AuxBIRs (input)* – A pointer to an array of elements of type BioAPI_INPUT_BIR. The number of elements of the array shall be *NumberOfAuxBIRs*. Each auxiliary BIR may be an intermediate, processed, score, decision, or biographic BIR, and any combination of such BIR types is allowed. The order of the elements is significant. If *NumberOfAuxBIRs* is zero, *AuxiliaryBIRs* may be a NULL pointer.

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

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

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

8.4.5.3 Return Value

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

8.4.5.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BIR
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_PURPOSE_NOT_SUPPORTED
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

8.4.6 BioAPI_VerifyMatch

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

8.4.6.1 Description

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

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

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

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

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

NOTE 1 Not all BSPs support return of payloads.

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

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit or BFP which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `SubBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

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

8.4.6.2 Parameters

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

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

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

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

— *AdaptedBIR (output/optional)* – A pointer to the handle of the adapted BIR. This parameter can be NULL if an adapted BIR is not desired. Not all BSPs support the adaptation of BIRs. The function may return a handle value of BioAPI_UNSUPPORTED_BIR_HANDLE to indicate that adaptation is not supported or a value of BioAPI_INVALID_BIR_HANDLE to indicate that adaptation was not possible.

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

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

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

8.4.6.3 Return Value

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

8.4.6.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BIR
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_BIR_NOT_FULLY_PROCESSED
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

8.4.7 BioAPI_VerifyMatchUsingAuxBIRs (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioAPI_VerifyMatchUsingAuxBIRs
    (BioAPI_HANDLE BSPHandle,
    BioAPI_FMR MaxFMRRequested,
    const BioAPI_INPUT_BIR *ProcessedBIR,
    const BioAPI_INPUT_BIR *ReferenceTemplate,
    uint32_t NumberOfAuxBIRs,
    const BioAPI_INPUT_BIR *AuxBIRs,
    BioAPI_BIR_HANDLE *AdaptedBIR,
    BioAPI_BOOL *Result,
    BioAPI_FMR *FMRAchieved,
    BioAPI_BIR_HANDLE *ResultBIR,
    BioAPI_DATA *Payload);
```

8.4.7.1 Description

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

The application shall request a maximum FMR value criterion (threshold) for a successful match.

In addition to the processed BIR and the reference template BIR, this function takes as input a list of auxiliary BIRs. Each auxiliary BIR in the list may be an intermediate, processed, score, decision, or biographic BIR.

The biometric service provider may modify the way it performs the verification operation based on the auxiliary BIRs provided as input and on the order in which they occur in the list. However, such modifications to the verification operation are not specified in this standard.

The main purposes of the auxiliary BIRs are:

a) to provide feedback to the biometric matching operation from prior processing, matching, fusion, or decision operations;

b) to modify the biometric matching operation based on statistical information (either specific to the subject, or specific to a population, or both).

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

The Boolean *Result* indicates whether verification was successful or not, and the *FMRAchieved* is a FMR value (score) indicating how closely the BIRs actually matched.

NOTE 1 See clause C.4 for information on the use of the FMR concept for normalized scoring and thresholding.

In addition to *Result*, this function returns a *ResultBIR*, which must be either a score BIR or a decision BIR containing a BDB whose format is chosen by the BSP. If a score BIR is returned, it must contain the score of the matching operation as specified by the particular "score" BDB format, along with any additional information specified for that BDB format. If a decision BIR is returned, it must contain the boolean result of the matching operation (whether verification was successful or not) as specified by the particular "decision" BDB format, along with any additional information specified for that BDB format.

It is recommended that BSP return a score BIR rather than a decision BIR whenever possible. While a decision BIR can only be used in decision-level fusion (thus preventing the use of score-level fusion), a score BIR can either be used in score-level fusion or passed as input to ***BioAPI_Decide*** to produce a decision BIR, which can then be used in decision-level fusion. Return of score BIRs results therefore in greater flexibility for applications that use biometric fusion.

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

NOTE 2 Not all BSPs support return of payloads.

NOTE 3 See clauses A.4.6.2.6 and C.5 for additional information regarding use of payloads.

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit or BFP which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

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

8.4.7.2 Parameters

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

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

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

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

— *NumberOfAuxBIRs (input)* – The number of elements (auxiliary BIRs) in the array pointed to by the parameter AuxBIRs.

— *AuxBIRs (input)* – A pointer to an array of elements of type BioAPI_INPUT_BIR. The number of elements of the array shall be NumberOfAuxBIRs. Each auxiliary BIR may be an intermediate, processed, score, decision, or biographic BIR, and any combination of such BIR types is allowed. The order of the elements is significant. If NumberOfAuxBIRs is zero, AuxBIRs may be a NULL pointer.

— *AdaptedBIR (output/optional)* – A pointer to the handle of the adapted BIR. This parameter can be NULL if an adapted BIR is not desired. Not all BSPs support the adaptation of BIRs. The function may return a handle value of BioAPI_UNSUPPORTED_BIR_HANDLE to indicate that adaptation is not supported or a value of BioAPI_INVALID_BIR_HANDLE to indicate that adaptation was not possible.

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

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

— *ResultBIR (output)* – A handle for the newly constructed "score" or "decision" BIR.

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

8.4.7.3 Return Value

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

8.4.7.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BIR
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_BIR_NOT_FULLY_PROCESSED
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

8.4.8 BioAPI_IdentifyMatch

```
BioAPI_RETURN BioAPI BioAPI_IdentifyMatch
    (BioAPI_HANDLE BSPHandle,
    BioAPI_FMR MaxFMRRequested,
    const BioAPI_INPUT_BIR *ProcessedBIR,
    const BioAPI_IDENTIFY_POPULATION *Population,
    uint32_t TotalNumberOfTemplates,
    BioAPI_BOOL Binning,
```



```
    uint32_t MaxNumberOfResults,
    uint32_t *NumberOfResults,
    BioAPI_CANDIDATE **Candidates,
    int32_t Timeout);
```

8.4.8.1 Description

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

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

b) input in an array of BIRs;

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

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

The function performs the following actions (in order):

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

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

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

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

NOTE 2 See clause C.4 for information on the use of the FMR concept for normalized scoring and thresholding.

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

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit which executes this function.

8.4.8.2 Parameters

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

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

— *ProcessedBIR (input)* – The BIR to be identified.

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

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

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

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

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

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

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

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

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

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

8.4.8.3 Return Value

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

8.4.8.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TIMEOUT_EXPIRED
BioAPIERR_NO_INPUT_BIRS
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_BIR_NOT_FULLY_PROCESSED
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_FUNCTION_FAILED
BioAPIERR_PRESET_BIR_DOES_NOT_EXIST
BioAPIERR_INVALID_DB_HANDLE
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

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

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

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

8.4.9 BioAPI_Decide (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioAPI_Decide
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_INPUT_BIR *ScoreBIR,
    uint32_t NumberOfAuxBIRs,
    const BioAPI_INPUT_BIR *AuxBIRs,
    BioAPI_BIR_HANDLE *DecisionBIR);
```

8.4.9.1 Description

This function processes a score BIR produced by a call to ***BioAPI_VerifyMatchUsingAuxBIRs*** or ***BioAPI_Fuse*** and builds a decision BIR.

In addition to the score BIR, this function takes as input a list of auxiliary BIRs. Each auxiliary BIR in the list may be an intermediate, processed, score, decision, or biographic BIR. The biometric service provider may modify the way it performs the decision operation based on the auxiliary BIRs provided as input and on the order in which they occur in the list. However, such modifications to the decision operation are not specified in this standard.

The main purposes of the auxiliary BIRs are:

a) to provide feedback to the biometric decision operation from prior processing, matching, fusion, or decision operations;

b) to modify the biometric decision operation based on statistical information (either specific to the subject, or specific to a population, or both).

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit or BFP which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

8.4.9.2 Parameters

— *BSPHandle (input)* – The handle of the attached BioAPI service provider.

— *ScoreBIR (input)* – The score BIR or its handle.

— *NumberOfAuxBIRs (input)* – The number of elements (auxiliary BIRs) in the array pointed to by the parameter AuxBIRs.

— *AuxBIRs (input)* – A pointer to an array of elements of type BioAPI_INPUT_BIR. The number of elements of the array shall be *NumberOfAuxBIRs*. Each auxiliary BIR may be an intermediate, processed, score, decision, or biographic BIR, and any combination of such BIR types is allowed. The order of the elements is significant. If *NumberOfAuxBIRs* is zero, *AuxBIRs* may be a NULL pointer.

— *DecisionBIR (output)* – A handle for the newly constructed decision BIR, or zero if the BSP does not support this function.

8.4.9.3 Return Value

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

8.4.9.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BIR
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_BIR_NOT_FULLY_PROCESSED
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

8.4.10 BioAPI_Fuse (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioAPI_Fuse
   (BioAPI_HANDLE BSPHandle,
   uint32_t NumberOfSourceBIRs,
   const BioAPI_INPUT_BIR *SourceBIRs,
   uint32_t NumberOfAuxBIRs,
   const BioAPI_INPUT_BIR *AuxBIRs,
   BioAPI_BIR_HANDLE *FusionBIR);
```

8.4.10.1 Description

This function performs a biometric fusion operation on the source BIRs provided as input, which shall be two or more BIRs of the same type (processed, score, or decision BIRs), and produces a fusion BIR. The order of the source BIRs is significant.

If the source BIRs are processed BIRs, the fusion BIR shall be a processed BIR. If the source BIRs are score BIRs, the fusion BIR shall be either a score BIR or a decision BIR. If the source BIRs are decision BIRs, the fusion BIR shall be a decision BIR.

In addition to the source BIRs, this function takes as input a list of auxiliary BIRs. Each auxiliary BIR in the list may be an intermediate, processed, score, decision, or biographic BIR. The biometric service provider may modify the way it performs the fusion operation based on the auxiliary BIRs provided as input and on the order in which they occur in the list. However, such modifications to the fusion operation are not specified in this standard.

The main purposes of the auxiliary BIRs are:

a) to provide feedback to the biometric fusion operation from prior processing, matching, fusion, or decision operations;

b) to modify the biometric fusion operation based on statistical information (either specific to the subject, or specific to a population, or both).

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit or BFP which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO

index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

8.4.10.2 Parameters

— *BSPHandle (input)* – The handle of the attached BioAPI service provider.

— *NumberOfSourceBIRs (input)* – The number of elements (source BIRs) in the array pointed to by the parameter SourceBIRs. This number shall be greater than or equal to two.

— *SourceBIRs (input)* – A pointer to an array of elements of type BioAPI_INPUT_BIR. The number of elements of the array shall be *NumberOfSourceBIRs*, and the BIRs shall all be of the same type (processed, score, or decision BIRs). The order of the elements is significant.

— *NumberOfAuxBIRs (input)* – The number of elements (auxiliary BIRs) in the array pointed to by the parameter AuxBIRs.

— *AuxBIRs (input)* – A pointer to an array of elements of type BioAPI_INPUT_BIR. The number of elements of the array shall be *NumberOfAuxBIRs*. Each auxiliary BIR may be an intermediate, processed, score, decision, or personal BIR, and any combination of such BIR types is allowed. The order of the elements is significant. If *NumberOfAuxBIRs* is zero, *AuxBIRs* may be a NULL pointer.

— *FusionBIR (output)* – A handle for the newly constructed fusion BIR, or zero if the BSP does not support fusion. The fusion BIR shall be a processed, score, or a decision BIR.

8.4.10.3 Return Value

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

8.4.10.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BIR
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_BIR_NOT_FULLY_PROCESSED
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

8.4.11 BioAPI_Enroll

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

8.4.11.1 Description

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

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

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

The BSP is responsible for providing the user interface associated with the enrollment operation as a default. The application may request control of the GUI "look-and-feel" by providing a GUI callback pointer in ***BioAPI_SetGUICallbacks***. See clause C.8 for additional explanation of user interface features.

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

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

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

8.4.11.2 Parameters

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

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

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

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

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

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

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

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

NOTE 1 Not all BSPs support storage of payloads.

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

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

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

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

8.4.11.3 Return Value

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

8.4.11.4 Errors

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

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

8.4.12 BioAPI_Verify

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

8.4.12.1 Description

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

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

NOTE 1 See clause C.4 for information on the use of the FMR concept for normalized scoring and thresholding.

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

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

NOTE 2 Not all BSPs support return of payloads.

NOTE 3 See clauses A.4.6.2.6 and C.5 for additional information regarding use of payloads.

The BSP is responsible for providing the user interface associated with the verify operation as a default. The application may request control of the GUI "look-and-feel" by providing a GUI callback pointer in ***BioAPI_SetGUICallbacks***. See clause C.8 for additional explanation of user interface features.

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

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Units or BFP which execute this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance(s). The current BPU IO index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance(s) and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

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

8.4.12.2 Parameters

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

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

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

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

NOTE Not all BSPs support the capture of specific Subtypes.

— *AdaptedBIR (output/optional)* – A pointer to the handle of the adapted BIR. This parameter can be NULL if an adapted BIR is not desired. Not all BSPs support the adaptation of BIRs. The function may return a handle value of BioAPI_UNSUPPORTED_BIR_HANDLE to indicate that adaptation is not supported or a value of BioAPI_INVALID_BIR_HANDLE to indicate that adaptation was not possible.

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

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

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

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

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

8.4.12.3 Return Value

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

8.4.12.4 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_UNABLE_TO_CAPTURE
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_TIMEOUT_EXPIRED
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_UNIT_IN_USE
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

8.4.13 BioAPI_Identify

```
BioAPI_RETURN BioAPI BioAPI_Identify
   (BioAPI_HANDLE BSPHandle,
   BioAPI_FMR MaxFMRRequested,
   BioAPI_BIR_SUBTYPE Subtype,
   const BioAPI_IDENTIFY_POPULATION *Population,
```

```
    uint32_t TotalNumberOfTemplates,
    BioAPI_BOOL Binning,
    uint32_t MaxNumberOfResults,
    uint32_t *NumberOfResults,
    BioAPI_CANDIDATE **Candidates,
    int32_t Timeout,
    BioAPI_BIR_HANDLE *AuditData);
```

8.4.13.1 Description

This function captures biometric data from the attached device (sensor unit), and compares it against a set of reference BIRs (the *Population)*.

The population that the match takes place against can be presented in one of two ways:

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

b) input in an array of BIRs;

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

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

The application shall request a maximum FMR value criterion for a successful match.

The function performs the following actions (in order):

i) captures a sample and processes it as appropriate;

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

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

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

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

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

By default, the BSP is responsible for providing the user interface associated with the identify operation. The application may, however, request control of the GUI "look-and-feel" by providing a GUI callback pointer in ***BioAPI_SetGUICallbacks***. See clause C.8 for additional explanation of user interface features.

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

The memory block returned by this BioAPI function call shall be freed by the application using ***BioAPI_Free*** (see clause 8.7.2). Output BIRs can be retrieved by a call to ***BioAPI_GetBIRFromHandle***, which releases the handle, or the handle can be released without retrieving the BIR through ***BioAPI_FreeBIRHandle***.

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit which executes this function.

8.4.13.2 Parameters

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

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

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

NOTE 1 Not all BSPs support the capture of specific Subtypes.

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

— *TotalNumberOfTemplates (input)* – Specifies the total number of templates stored by the application in the population. A value of NULL is used if not specified.

NOTE 2 If the total population is distributed over several databases/partitions, then the total size of the population will be greater than the population seen by the BSP. The BSP can map the *FARRequested* to its internal matching threshold based on this total population size.

— *Binning (input/optional)* – A Boolean indicating whether binning is on or off.

NOTE 3 Binning is a search optimization technique that the BSP can employ. It is based on searching a subset of the population according to the intrinsic characteristics of the biometric data. While it can improve the speed of the Match operation, it can also increase the probability of missing a candidate (due to the possibility of mis-binning and as a result, searching a bin which should, but does not, contain the matching BIR).

NOTE 4 Additional information regarding binning is located in clause A.4.6.2.10.

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

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

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

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

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

8.4.13.3 Return Value

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

8.4.13.4 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_UNABLE_TO_CAPTURE
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_TIMEOUT_EXPIRED
BioAPIERR_NO_INPUT_BIRS
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_UNIT_IN_USE
BioAPIERR_PRESET_BIR_DOES_NOT_EXIST
BioAPIERR_INVALID_DB_HANDLE
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
```

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

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

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

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

8.4.14 BioAPI_Import

```
BioAPI_RETURN BioAPI BioAPI_Import
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_DATA *InputData,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *InputFormat,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_BIR_HANDLE *ConstructedBIR);
```

8.4.14.1 Description

This function passes raw biometric data obtained by a biometric application by any means, and requests the specified BSP attach session to construct a BIR for the purpose specified. *InputData* identifies the memory buffer containing the raw biometric data, while *InputFormat* identifies the form of the raw biometric data. The *InputFormats* that a particular BSP will be prepared to accept are determined by the BSP (see the error BioAPIERR_UNSUPPORTED_FORMAT). The function returns a handle to the *ConstructedBIR*. If the application needs to acquire the BIR either to store it in a database or to send it to a server, the application can retrieve the data with the ***BioAPI_GetBIRFromHandle*** function, or store it directly using ***BioAPI_DbStoreBIR***.

The output *ConstructedBIR* can be retrieved by a call to ***BioAPI_GetBIRFromHandle***, which releases the handle, or the handle can be released without retrieving the BIR through ***BioAPI_FreeBIRHandle***.

8.4.14.2 Parameters

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

— *InputData (input)* – A pointer to image/stream data to import into a Processed BIR. The image/stream conforms to the standard identified by *InputFormat*.

— *InputFormat (input)* – The format of the *InputData*.

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

— *Purpose (input)* – A value indicating the purpose of the *ConstructedBIR*.

— *ConstructedBIR (output)* – A handle to a BIR constructed from the imported biometric data. This BIR may be either an Intermediate or Processed BIR (to be indicated in the header).

8.4.14.3 Return Value

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

8.4.14.4 Errors

```
BioAPIERR_UNSUPPORTED_FORMAT
BioAPIERR_UNABLE_TO_IMPORT
BioAPIERR_TOO_MANY_HANDLES
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_PURPOSE_NOT_SUPPORTED
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
```

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

8.4.15 BioAPI_Export (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioAPI_Export
   (BioAPI_HANDLE BSPHandle,
   BioAPI_BIR_HANDLE Handle,
   const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputBDBFormat,
   const BioAPI_BIR_SECURITY_BLOCK_FORMAT *OutputSBFormat,
   BioAPI_BIR *BIR);
```

8.4.15.1 Description

This function returns the BIR of the BDB format identified by OutputBDBFormat and the SB format identified by OutputSBFormat, associated with a BIR handle and stored in BioAPI_ASN1_BIR form, returned by a BSP.

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit which executes this function.

8.4.15.2 Parameters

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

— *Handle (input)* – The handle of the BIR to be exported.

— *OutputBDBFormat (input)* – Specifies which BDB format to use for the returned BIR, if the BSP supports more than one format.

— *OutputSBFormat (input/optional)* – Specifies which SB format to use for the returned BIR, if the BSP supports more than one format. A NULL pointer value indicates that SB is not used for the returned BIR.

— *BIR (output)* – Pointer to the exported BIR.

8.4.15.3 Return Value

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

8.4.15.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_INVALID_BSP_HANDLE
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
```

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

8.4.16 BioAPI_PresetIdentifyPopulation

```
BioAPI_RETURN BioAPI BioAPI_PresetIdentifyPopulation
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_IDENTIFY_POPULATION *Population);
```

8.4.16.1 Description

This function provides the population of BIRs to the BSP, as specified in the *BSPHandle*. The enrollment population to be matched against can be presented in one of two ways:

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

b) input in an array of BIRs.

The BSP allocates a memory block and transfers the population of BIRs to the memory block with a data format (not standardized) which is supported by the currently attached matching algorithm BioAPI Unit. After this function is called successfully, an application can call ***BioAPI_Identify*** or ***BioAPI_IdentifyMatch,*** specifying BioAPI_PRESET_ARRAY_TYPE in the BioAPI_IDENTIFY_POPULATION structure. The BSP keeps this memory block until another ***BioAPI_PresetIdentifyPopulation*** or ***BioAPI_BSPDetach*** is called.

8.4.16.2 Parameters

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

— *Population (input)* – The population of BIRs against which the identification is performed (by this BSP).

8.4.16.3 Return Value

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

8.4.16.4 Errors

```
BioAPIERR_INVALID_BIR_HANDLE
BioAPIERR_BIR_SIGNATURE_FAILURE
BioAPIERR_NO_INPUT_BIRS
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INCONSISTENT_PURPOSE
BioAPIERR_BIR_NOT_FULLY_PROCESSED
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_QUALITY_ERROR
BioAPIERR_FUNCTION_FAILED
```

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

NOTE Depending on the size of the database to be transferred and the degree of transformation required, this operation could take a significant amount of time to perform.

8.5 Database Operations

The database operations provide application access to BIR databases on archive BioAPI Units (either directly managed by a BSP or implemented via a BFP).

NOTE 1 See clause C.7 for an explanation of the context for and usage of the database functions.

NOTE 2 Authorization mechanisms to control a biometric application's right to create or delete or write to a database are outside the scope of this document (but may be addressed in 19784-2, Biometric Archive Function Provider Interface).

8.5.1 BioAPI_DbOpen

```
BioAPI_RETURN BioAPI BioAPI_DbOpen
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_UUID *DbUuid,
    BioAPI_DB_ACCESS_TYPE AccessRequest,
    BioAPI_DB_HANDLE *DbHandle,
    BioAPI_DB_MARKER_HANDLE *MarkerHandle);
```

8.5.1.1 Description

This function opens a BIR database maintained by the currently attached archive of the identified BSP invocation, using the access mode specified by the *AccessRequest*. A new marker is created and set to point to the first record in the BIR database, and a handle for that marker is returned.

Some BSPs may only support a single BIR database or may have a preferred database. The application can allow the BSP to select the BIR database to open by using a NULL pointer value for the database UUID parameter.

Every call to this function should be followed at some point by a call to ***BioAPI_DbFreeMarker***, to allow the marker resources to be released. Applications should be aware that every call to ***BioAPI_DbOpen*** results in the allocation of some resources that cannot be automatically freed (otherwise it would be impossible to iterate through the database using the returned marker handle). If the application doesn't want to iterate through the BIR database, it should call ***BioAPI_DbFreeMarker*** immediately, otherwise it should do so after terminating the iteration.

8.5.1.2 Parameters

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

— *DbUuid (input)* – A pointer to a UUID identifying the BIR database to be opened. If this value is set to NULL, then the BSP shall select the BIR database to open.

— *AccessRequest (input)* – An indicator of the requested access mode for the BIR database, BioAPI_DB_ACCESS_READ or BioAPI_DB_ACCESS_WRITE. On a single system, only one application at a time is allowed to open a target database in BioAPI_DB_ACCESS_WRITE mode.

— *DbHandle (output)* – A handle for the opened BIR database. The value will be set to BioAPI_DB_INVALID_HANDLE if the function fails.

— *MarkerHandle (output)* – A marker handle that can be subsequently used by the application to iterate through the BIR database from the first record.

8.5.1.3 Return Value

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

8.5.1.4 Errors

```
BioAPIERR_UNABLE_TO_OPEN_DATABASE
BioAPIERR_DATABASE_IS_OPEN
BioAPIERR_DATABASE_IS_LOCKED
BioAPIERR_DATABASE_DOES_NOT_EXIST
BioAPIERR_INVALID_UUID
BioAPIERR_INVALID_ACCESS_REQUEST
BioAPIERR_DATABASE_IS_CORRUPT
```

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

8.5.2 BioAPI_DbClose

```
BioAPI_RETURN BioAPI BioAPI_DbClose
    (BioAPI_HANDLE BSPHandle,
   BioAPI_DB_HANDLE DbHandle);
```

8.5.2.1 Description

This function closes an open BIR database. All markers currently set for records in the database are freed and their handles become invalid.

NOTE A database opened in BioAPI_DB_ACCESS_WRITE mode can be damaged if it is left unclosed.

8.5.2.2 Parameters

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

— *DbHandle (input)* – The DB handle for an open BIR database controlled by the BSP. This specifies the open database to be closed.

8.5.2.3 Return Value

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

8.5.2.4 Errors

```
BioAPIERR_UNABLE_TO_CLOSE_DATABASE
BioAPIERR_INVALID_DB_HANDLE
BioAPIERR_DATABASE_IS_CORRUPT
```

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

8.5.3 BioAPI_DbCreate

```
BioAPI_RETURN BioAPI BioAPI_DbCreate
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_UUID *DbUuid,
    uint32_t NumberOfRecords,
    BioAPI_DB_ACCESS_TYPE AccessRequest,
    BioAPI_DB_HANDLE *DbHandle);
```

8.5.3.1 Description

This function creates and opens a new BIR database on the currently attached archive unit of the identified BSP invocation. The identification of the new database is specified by the input parameter *DbUuid* which shall be created by the biometric application, and shall be distinct from any current database UUID supported by that archive unit, whether currently open or not. The newly created BIR database is opened under the specified access mode.

NOTE This function is used create a new BIR database. It does not transport any information to the archive unit about the new database except its UUID and access conditions. There are archives which are only able to deal with static database sizes or where much higher effort will be needed to manage a database with dynamic size (e.g., smartcards storing templates in transparent or structured files, which might have a static size dependent on the characteristics of the smartcard operating system). Archives can need size information to create a new BIR database. Because the calling application might not have the knowledge about typical or maximum template sizes in bytes, the number of records to be stored at maximum in the database will be provided. Archives which are able to deal with database sizes dynamically can ignore the *NumberOfRecords* parameter.

8.5.3.2 Parameters

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

— *DbUuid (input)* – A pointer to a UUID that will identify the BIR database to be created.

— *NumberOfRecords (input)* – The maximum number of records to be stored at in the BIR database.

— *AccessRequest (input)* – An indicator of the requested access mode for the BIR database, such as **read** or **write**. On a single system, only one application at a time is allowed to open a target database in BioAPI_DB_ACCESS_WRITE mode.

— *DbHandle (output)* – The handle to the newly created and open database. The value will be set to BioAPI_DB_INVALID_HANDLE if the function fails.

8.5.3.3 Return Value

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

8.5.3.4 Errors

```
BioAPIERR_UNABLE_TO_CREATE_DATABASE
BioAPIERR_DATABASE_ALREADY_EXISTS
BioAPIERR_INVALID_UUID
BioAPIERR_INVALID_ACCESS_REQUEST
```

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

8.5.4 BioAPI_DbDelete

```
BioAPI_RETURN BioAPI BioAPI_DbDelete
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_UUID *DbUuid);
```

8.5.4.1 Description

This function deletes all records from the specified BIR database and removes all state information associated with that database.

8.5.4.2 Parameters

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

— *DbUuid (input)* – A pointer to a UUID identifying the BIR database to be deleted.

8.5.4.3 Return Value

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

8.5.4.4 Errors

```
BioAPIERR_DATABASE_IS_OPEN
BioAPIERR_INVALID_UUID
```

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

8.5.5 BioAPI_DbSetMarker

```
BioAPI_RETURN BioAPI BioAPI_DbSetMarker
    (BioAPI_HANDLE BSPHandle,
    BioAPI_DB_HANDLE DbHandle,
    const BioAPI_UUID *KeyValue,
    BioAPI_DB_MARKER_HANDLE MarkerHandle);
```

8.5.5.1 Description

The marker identified by the *MarkerHandle* is set to point to the record indicated by the *KeyValue* in the BIR database identified by the *DbHandle*. A NULL value will set the marker to the first record in the database.

NOTE When an error occurs, the position of the marker does not change.

8.5.5.2 Parameters

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

— *DbHandle (input)* – A handle to the open BIR database on the attached archive BioAPI Unit.

— *KeyValue (input)* – The key into the database of the BIR to which the marker is to be set.

— *MarkerHandle (input)* – The handle of the marker that is to be set. This marker handle can be subsequently used by the application to iterate through the BIR database from the new position.

8.5.5.3 Return Value

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

8.5.5.4 Errors

```
BioAPIERR_INVALID_DB_HANDLE
BioAPIERR_RECORD_NOT_FOUND
```

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

8.5.6 BioAPI_DbFreeMarker

```
BioAPI_RETURN BioAPI BioAPI_DbFreeMarker
    (BioAPI_HANDLE BSPHandle,
    BioAPI_DB_MARKER_HANDLE MarkerHandle);
```


8.5.6.1 Description

Frees memory and resources associated with the specified marker and invalidates the *MarkerHandle*.

8.5.6.2 Parameters

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

— *MarkerHandle (input)* – The handle of the BIR database marker to be freed.

8.5.6.3 Return Value

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

8.5.6.4 Errors

`BioAPIERR_MARKER_HANDLE_IS_INVALID`

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

8.5.7 BioAPI_DbStoreBIR

```
BioAPI_RETURN BioAPI BioAPI_DbStoreBIR
   (BioAPI_HANDLE BSPHandle,
   const BioAPI_INPUT_BIR *BIRToStore,
   BioAPI_DB_HANDLE DbHandle,
   BioAPI_UUID *BirUuid);
```

8.5.7.1 Description

The BIR identified by the *BIRToStore* parameter is stored in the open BIR database identified by the *DbHandle* parameter. If the *BIRToStore* is identified by a BIR Handle, the input BIR Handle is freed. If the *BIRToStore* is identified by a database key value, the BIR is retrieved and stored in (copied to) the open database. A new UUID is assigned to the new BIR in the database, and this UUID can be used as a key value to access the BIR later.

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

8.5.7.2 Parameters

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

— *BIRToStore (input)* – The BIR to be stored in the open BIR database (either the BIR, or its handle, or the index to it in another open database).

— *DbHandle (input)* – The handle to the open BIR database.

— *BirUuid (output)* – A UUID that uniquely identifies the new BIR in the BIR database. This UUID cannot be changed. To associate a different BIR with the user, it is necessary to delete the old one, store a

new one in the BIR database, and then replace the old UUID with the new one in the application account database. The BIR is added to the tail end of the database.

8.5.7.3 Return Value

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

8.5.7.4 Errors

```
BioAPIERR_INVALID_DB_HANDLE
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

8.5.8 BioAPI_DbGetBIR

```
BioAPI_RETURN BioAPI BioAPI_DbGetBIR
    (BioAPI_HANDLE BSPHandle,
    BioAPI_DB_HANDLE DbHandle,
    const BioAPI_UUID *KeyValue,
    BioAPI_BIR_HANDLE *RetrievedBIR,
    BioAPI_DB_MARKER_HANDLE *MarkerHandle);
```

8.5.8.1 Description

The BIR identified by the *KeyValue* parameter in the open BIR database identified by the *DbHandle* parameter is retrieved. The BIR is copied into BSP storage and a handle to it is returned. A marker is created and set to point to the record following the retrieved BIR in the BIR database (or to the first record in the database if the retrieved BIR is the last), and a handle for the marker is returned.

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

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

8.5.8.2 Parameters

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

— *DbHandle (input)* – The handle to the open BIR database.

— *KeyValue (input)* – The key into the database of the BIR to retrieve.

— *RetrievedBIR (output)* – A handle to the retrieved BIR.

— *MarkerHandle (output)* – A marker handle that can be subsequently used to iterate through the BIR database from the record following the retrieved BIR.

8.5.8.3 Return Value

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

8.5.8.4 Errors

```
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_INVALID_DB_HANDLE
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

8.5.9 BioAPI_DbGetNextBIR

```
BioAPI_RETURN BioAPI BioAPI_DbGetNextBIR
    (BioAPI_HANDLE BSPHandle,
    BioAPI_DB_HANDLE DbHandle,
    BioAPI_DB_MARKER_HANDLE MarkerHandle,
    BioAPI_BIR_HANDLE *RetrievedBIR,
    BioAPI_UUID *BirUuid);
```

8.5.9.1 Description

The BIR identified by the *MarkerHandle* parameter is retrieved. The BIR is copied into BSP storage and a handle to it is returned, and a pointer to the UUID that uniquely identifies the BIR in the database is returned. The marker is updated to point to the next record in the database.

NOTE If there are no more records left in the BIR database, the marker points to an invalid position.

If ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) corresponding to the BioAPI Unit which executes this function. If ACBio instance generation is requested, the challenge given as *Challenge* in the parameter *ACBioParameters* of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) shall be set into the field `controlValue` of type `ACBioContentInformation` in the ACBio instance. The current BPU IO index for the BSP Unit shall be set into the field `bpuIOIndex` of `bpuOuputExecutionInformationList` in the `biometricProcess` of `ACBioContentInformation` in the ACBio instance and also set into `bpuIOIndex` in `subBlockForACBio` in the security block. After that the current BPU IO index shall be incremented. The BIR shall be stored in type BioAPI_ASN1_BIR.

8.5.9.2 Parameters

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

— *DbHandle (input)* – The handle to the open BIR database.

— *MarkerHandle (input/output)* – A marker handle indicating which record to retrieve.

— *RetrievedBIR (output)* – A handle to the retrieved BIR.

— *BirUuid (output)* – The UUID that uniquely identifies the retrieved BIR in the BIR database.

8.5.9.3 Return Value

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

8.5.9.4 Errors

```
BioAPIERR_END_OF_DATABASE
BioAPIERR_MARKER_HANDLE_IS_INVALID
BioAPIERR_INVALID_DB_HANDLE
BioAPIERR_SECURITY_ENCRYPTION_FAILURE
BioAPIERR_SECURITY_DECRYPTION_FAILURE
BioAPIERR_SECURITY_MAC_GENERATION_FAILURE
BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE
BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE
```

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

8.5.10 BioAPI_DbDeleteBIR

```
BioAPI_RETURN BioAPI BioAPI_DbDeleteBIR
    (BioAPI_HANDLE BSPHandle,
    BioAPI_DB_HANDLE DbHandle,
    const BioAPI_UUID *KeyValue);
```

8.5.10.1 Description

The BIR identified by the *KeyValue* parameter in the open BIR database identified by the *DbHandle* parameter is deleted from the database. If there is a marker set to the deleted BIR, then:

a) if that BIR was not the last BIR in the database, then the marker is moved to the next sequential BIR;

b) otherwise, the marker points to an invalid position. However, its marker handle remains valid and can be used in a subsequent call to *BioAPI_DbSetMarker* to set the marker to point to a different record.

8.5.10.2 Parameters

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

— *DbHandle (input)* – The handle to the open BIR database.

— *KeyValue (input)* – The UUID of the BIR to be deleted.

8.5.10.3 Return Value

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

8.5.10.4 Errors

```
BioAPIERR_RECORD_NOT_FOUND
BioAPIERR_INVALID_DB_HANDLE
```

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

8.6 BioAPI Unit operations

8.6.1 BioAPI_SetPowerMode

```
BioAPI_RETURN BioAPI BioAPI_SetPowerMode
    (BioAPI_HANDLE BSPHandle,
    BioAPI_UNIT_ID UnitId,
    BioAPI_POWER_MODE PowerMode);
```


8.6.1.1 Description

This function sets the currently attached BioAPI Unit of the referenced BSP attach session to the requested power mode if the BioAPI Unit supports it.

8.6.1.2 Parameters

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

— *UnitId (input)* – The ID of the BioAPI Unit for which the power mode is to be set. The BioAPI_DONT_CARE option is not valid for this function.

— *PowerMode (input)* – A 32-bit value indicating the power mode to set the BioAPI Unit to.

8.6.1.3 Return Value

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

8.6.1.4 Errors

```
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_UNIT_NOT_INSERTED
BioAPIERR_UNIT_IN_USE
```

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

8.6.2 BioAPI_SetIndicatorStatus

```
BioAPI_RETURN BioAPI BioAPI_SetIndicatorStatus
    (BioAPI_HANDLE BSPHandle,
    BioAPI_UNIT_ID UnitId,
    BioAPI_INDICATOR_STATUS IndicatorStatus);
```

8.6.2.1 Description

This function sets the selected BioAPI Unit to the requested indicator status if the BioAPI Unit supports it. After BioAPI_INDICATOR_ACCEPT or BioAPI_INDICATOR_REJECT is set in the *IndicatorStatus* parameter, the status will not be changed until the application sets another value.

8.6.2.2 Parameters

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

— *UnitId (input)* – The ID of the BioAPI Unit for which the indicator status is to be set. The BioAPI_DONT_CARE option is not valid for this function.

— *IndicatorStatus (input)* – A value to which to set the indicator status of the BioAPI Unit.

8.6.2.3 Return Value

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

8.6.2.4 Errors

```
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_UNIT_NOT_INSERTED
BioAPIERR_UNIT_IN_USE
```

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

8.6.3 BioAPI_GetIndicatorStatus

```
BioAPI_RETURN BioAPI BioAPI_GetIndicatorStatus
   (BioAPI_HANDLE BSPHandle,
   BioAPI_UNIT_ID UnitId,
   BioAPI_INDICATOR_STATUS *IndicatorStatus);
```

8.6.3.1 Description

This function returns the indicator status of the BioAPI Unit if that BioAPI Unit supports it.

8.6.3.2 Parameters

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

— *UnitId (input)* – The ID of the BioAPI Unit for which the indicator status is to be obtained. The BioAPI_DONT_CARE option is not valid for this function.

— *IndicatorStatus (output)* – A value for the indicator status of the BioAPI Unit.

8.6.3.3 Return Value

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

8.6.3.4 Errors

```
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_UNIT_NOT_INSERTED
BioAPIERR_UNIT_IN_USE
```

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

8.6.4 BioAPI_CalibrateSensor

```
BioAPI_RETURN BioAPI BioAPI_CalibrateSensor
       (BioAPI_HANDLE BSPHandle,
       int32_t Timeout);
```

8.6.4.1 Description

This function performs a calibration of the attached sensor BioAPI Unit if that sensor unit supports it.

8.6.4.2 Parameters

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

— *Timeout (input)* – An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error. This value can be any positive number. A '-1' value means the BSP's default calibration timeout value will be used.

8.6.4.3 Return Value

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

8.6.4.4 Errors

```
BioAPIERR_FUNCTION_NOT_SUPPORTED
BioAPIERR_UNIT_IN_USE
BioAPIERR_INVALID_UNIT_ID
BioAPIERR_UNIT_NOT_INSERTED
BioAPIERR_CALIBRATION_NOT_SUCCESSFUL
BioAPIERR_TIMEOUT_EXPIRED
```

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

8.7 Utility Functions

8.7.1 BioAPI_Cancel

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

8.7.1.1 Description

This function shall cancel any presently blocked calls associated with *BSPHandle*. The function shall not return until all blocking calls have been cancelled.

This function may be called from any thread.

This function shall not be called in the callback function.

8.7.1.2 Parameters

— *BSPHandle (input)* – The handle of the attached BSP.

8.7.1.3 Return Value

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

8.7.1.4 Errors

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

8.7.2 BioAPI_Free

```
BioAPI_RETURN BioAPI BioAPI_Free
    (void* Ptr);
```

8.7.2.1 Description

This function causes the memory block pointed to by *Ptr* to be deallocated. If *Ptr* is NULL, no action occurs. Otherwise, if *Ptr* does not match a pointer earlier returned by the BioAPI functions, or if the memory block already has been deallocated by a call to ***BioAPI_Free***, the behavior is undefined.

There are some BioSPI functions in which the BSP allocates a memory block which is to be freed by the Framework by calling ***BioSPI_Free***. Whenever the Framework passes one such memory block up to the application, it makes the application responsible for deallocating the memory block. In such cases, the application will simply call ***BioAPI_Free***, and the Framework (on reception of that call) must either free the memory block or call ***BioSPI_Free*** on the appropriate attach session of the appropriate BSP.

NOTE The Framework can be implemented in such a way that the BSP allocated pointers are not passed to the application. The Framework will in this case move the data returned by the BSP to newly allocated memory, and will call ***BioSPI_Free*** after copying the memory, but before returning back to the application. However, such differences in the behaviour of the Framework are not visible to the application.

In other cases, the Framework itself allocates a memory block and passes it to the application. In such cases, when the application calls ***BioAPI_Free***, the Framework must simply deallocate the memory block, and must not call ***BioSPI_Free*** on any BSP.

This mechanism requires that the Framework keep track of which memory blocks it has allocated itself and which memory blocks it has received from a BSP. For the latter, it must keep track of the BSP and the attach session from which the Framework has received the memory block.

8.7.2.2 Parameters

— *Ptr (input)* – A pointer to the memory to free.

8.7.2.3 Return Value

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

8.7.2.4 Errors

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

9 BioAPI Service Provider Interface

9.1 Summary

The service provider interface (SPI) is the programming interface that a BSP shall present in order to interwork with the BioAPI Framework, or with one or more applications in a framework-free BioAPI system.

In general, for a full BioAPI system, the SPI is a one-to-one mapping of a function call from a biometric application to the BioAPI Framework (using the BioAPI API specified in Clause 8) down to an attach session. The BioAPI Framework routes API calls down to the corresponding SPI of the specified BSP (as identified by the BSP handle parameter in the API function). The following conventions have been adopted in this clause:

a) where an SPI function has parameters that are identical to those of an API function of the same name (apart from the omission of the BSP handle), all that is presented in this clause is the signature of the SPI function, with no further explanation. The reader can then refer to the corresponding API definition for the details. (Note that reference to such text may be needed when building a BSP for use in a framework-free system.);

b) where the parameters of an SPI function differ from those of an API function of the same name, a full definition of the SPI function is presented;

c) Not all API functions have a corresponding SPI function; those functions that do not are handled completely by the BioAPI Framework.

9.2 Type Definitions for Biometric Service Providers

9.2.1 BioSPI_EventHandler

An application should not assume that this function is available when operating with a BSP designed for a framework-free system, as such a BSP is not required to support this function.

The ***BioAPI_EventHandler*** function prototype (clause 7.38) is used to define the event handler interface that enables the BioAPI Framework to receive asynchronous notification of events of type BioAPI_EVENT from a BSP. Example events include insertion or removal of a BioAPI Unit or a fault detection.

The address of a ***BioSPI_EventHandler*** function is passed to the BSP during ***BioSPI_BSPLoad***. This is the single event handler that a BSP should call to notify the BioAPI Framework of event types that occur in a loaded BSP.

There is only one event handler for any given BSP that is loaded as a result of a ***BioAPI_BSPLoad*** from one or more applications. It is the responsibility of the BioAPI Framework to notify the (possibly multiple) biometric application event handlers for applications that have loaded the BSP in which an event occurs. Where an insert event has been notified to the Framework before a given biometric application loads the BSP (but after another biometric application has loaded it), the Framework remembers that event and notifies that biometric application's event handler immediately after the load by the biometric application.

The BioAPI Framework forwards events to the biometric application that invoked the corresponding ***BioAPI_BSPLoad*** function. The handler specified in ***BioSPI_EventHandler*** can (but need not) be invoked multiple times in response to a single event.

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

9.2.1.1 Definitions

— *BSPUuid (input)* – The UUID of the biometric service provider raising the event.

— *UnitID (input)* – The ID of the BioAPI Unit associated with the event.

— *UnitSchema (input)* – A pointer to the unit schema of the BioAPI Unit associated with the event.

— *EventType (input)* – The BioAPI_EVENT that has occurred.

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

When the framework receives (from a BSP) a call to an event handler that carries a unit schema, the framework shall not call ***BioSPI_Free*** to free the memory block containing the unit schema or a memory block pointed to by any of its members.

9.2.2 BioSPI_BFP_ENUMERATION_HANDLER

An application should not assume that this function is available when operating with a BSP designed for a framework-free system, as such a BSP is not required to support this function.

This is a callback that the BioAPI Framework exposes to the BSPs to enable them to obtain information about the installed BFPs. This callback is similar to the BioAPI function ***BioAPI_EnumBFPs*** (see clause 8.1.11), but unlike that function, it is not part of the BioAPI API, and thus it is not exposed to applications. Conversely, a BSP can use this callback in order to obtain the same information that an application would obtain by calling the BioAPI function ***BioAPI_EnumBFPs***.

The address of the BioSPI_BFP_ENUMERATION_HANDLER callback is provided by the Framework to a BSP as an input parameter of the ***BioSPI_BSPLoad*** function.

9.2.2.1 Definition

```
typedef BioAPI_RETURN (*BioSPI_BFP_ENUMERATION_HANDLER)
   (BioAPI_BFP_SCHEMA **BFPSchemaArray,
   uint32_t *NumberOfElements);
```

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

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

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

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

The memory block containing the array shall be freed by the BSP via a callback to the framework's memory deallocation handler (see clause 9.2.3) when it is no longer needed by the BSP.

The memory blocks pointed to by the *Path* and *BFPProperty* members within each element of the array shall also be freed by the BSP via a callback to the framework's memory deallocation handler (see clause 9.2.3) when they are no longer needed by the application.

9.2.2.2 Parameters

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

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

9.2.2.3 Return Value

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

9.2.2.4 Errors

See BioAPI Error Handling (Clause 11).

9.2.3 BioSPI_MEMORY_FREE_HANDLER

An application should not assume that this function is available when operating with a BSP designed for a framework-free system, as such a BSP is not required to support this function.

This is also called a memory deallocation handler. It is a callback that the BioAPI Framework exposes to the BSPs to enable them to request the deallocation of a memory block that was allocated by the framework to return data to the BSP during a prior callback. Such allocations occur when a BSP calls the framework's BFP enumeration handler (see clause 9.2.2). A memory deallocation handler is similar to the BioAPI function ***BioAPI_Free*** (see clause 8.7.2), but unlike that function, it is not part of the BioAPI API, and thus it is not exposed to applications.

The address of the BioSPI_MEMORY_FREE_HANDLER callback is provided by the Framework to a BSP as an input parameter of the ***BioSPI_BSPLoad*** function.

9.2.3.1 Definition

```
typedef BioAPI_RETURN (*BioSPI_MEMORY_FREE_HANDLER)
    (void* Ptr);
```

This callback causes the memory block pointed to by *Ptr* to be deallocated by the framework. If *Ptr* is NULL, no action occurs. Otherwise, if *Ptr* does not match a pointer earlier returned by a callback, or if

the memory block already has been deallocated by an earlier call to the memory deallocation handler, the behavior is undefined.

NOTE Unlike in the function ***BioAPI_Free***, no requests made to a memory deallocation handler result in a call to ***BioSPI_Free***.

9.2.3.2 Parameters

— *Ptr (input)* – A pointer to the memory block to free.

9.2.3.3 Return Value

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

9.2.3.4 Errors

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

9.2.4 BioSPI_GUI_PROGRESS_EVENT_HANDLER (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

9.2.4.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioSPI_GUI_PROGRESS_EVENT_HANDLER)
   (const BioAPI_UUID *BSPUuid,
   BioAPI_UNIT_ID UnitID,
   const BioAPI_HANDLE *BSPHandle,
   BioAPI_GUI_OPERATION Operation,
   BioAPI_GUI_SUBOPERATION Suboperation,
   BioAPI_BIR_PURPOSE Purpose,
   BioAPI_GUI_MOMENT Moment,
   uint8_t SuboperationProgress,
   const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
   const uint8_t *Text,
   BioAPI_GUI_RESPONSE *Response);
```

9.2.4.2 Description

This is a function pointer type for a framework's GUI event handler function that is to handle GUI progress event notification callbacks coming from a BSP. In order to receive GUI progress event notifications, the framework shall register a callback function of type BioSPI_GUI_PROGRESS_EVENT_HANDLER by providing the callback address of the function in a call to ***BioSPI_SubscribeToGUIEvents*** (see 9.3.3.3).

The BSP makes a callback to a framework function of this type in order to notify to the framework (and ultimately to the application) a GUI progress event generated while executing a BioSPI function.

A BSP can generate GUI progress events only during the execution of a call to ***BioSPI_Capture***, ***BioSPI_Process***, ***BioSPI_CreateTemplate***, ***BioSPI_VerifyMatch***, ***BioSPI_IdentifyMatch***, ***BioSPI_Verify***, ***BioSPI_Identify***, or ***BioSPI_Enroll***. They can be generated at any time, even multiple times, during any suboperation.

9.2.4.3 Parameters

The parameters of this function pointer type are the same as those of ***BioAPI_GUI_PROGRESS_EVENT_HANDLER***, except for the absence of the context address parameter.

NOTE See also C.8.

9.2.4.4 Return Value

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

9.2.4.5 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED
```

9.2.5 BioSPI_GUI_SELECT_EVENT_HANDLER (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

9.2.5.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioSPI_GUI_SELECT_EVENT_HANDLER)
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    BioAPI_GUI_ENROLL_TYPE   EnrollType,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
    int32_t MaxNumEnrollSamples,
    BioAPI_BIR_SUBTYPE_MASK SelectableInstances,
    BioAPI_BIR_SUBTYPE_MASK *SelectedInstances,
    BioAPI_BIR_SUBTYPE_MASK CapturedInstances,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response);
```

9.2.5.2 Description

This is a function pointer type for a framework's GUI event handler function that is to handle GUI select event notification callbacks coming from a BSP. In order to receive GUI select event notifications, the framework shall register a callback function of type BioSPI_GUI_SELECT_EVENT_HANDLER by providing the callback address of the function in a call to ***BioSPI_SubscribeToGUIEvents*** (see 9.3.3.3).

The BSP makes a callback to a framework function of this type in order to notify to the framework (and ultimately to the application) each GUI select event generated while executing a BioSPI function.

A BSP can generate GUI select events only during the execution of a call to ***BioSPI_Capture***, ***BioSPI_Verify***, ***BioSPI_Identify***, or ***BioSPI_Enroll***. A GUI select event with the moment value BioAPI_GUI_MOMENT_BEFORE_START is generated before the start of each suboperation cycle, and a GUI select event with the moment value BioAPI_GUI_MOMENT_AFTER_END is generated after the end of each suboperation cycle (see 7.47).

9.2.5.3 Parameters

The parameters of this function pointer type are the same as those of ***BioAPI_GUI_SELECT_EVENT_HANDLER***, except for the absence of the *context address* parameter.

NOTE See also C.8.

9.2.5.4 Return Value

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

9.2.5.5 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED
```


9.2.6 BioSPI_GUI_STATE_EVENT_HANDLER (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

9.2.6.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioSPI_GUI_STATE_EVENT_HANDLER)
   (const BioAPI_UUID *BSPUuid,
   BioAPI_UNIT_ID UnitID,
   const BioAPI_HANDLE *BSPHandle,
   BioAPI_GUI_OPERATION Operation,
   BioAPI_GUI_SUBOPERATION Suboperation,
   BioAPI_BIR_PURPOSE Purpose,
   BioAPI_GUI_MOMENT Moment,
   BioAPI_RETURN ResultCode,
   int32_t EnrollSampleIndex,
   const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
   const uint8_t *Text,
   BioAPI_GUI_RESPONSE *Response,
   int32_t *EnrollSampleIndexToRecapture);
```

9.2.6.2 Description

This is a function pointer type for a framework's GUI event handler function that is to handle GUI state event notification callbacks coming from a BSP. In order to receive GUI state event notifications, the framework shall register a callback function of type BioSPI_GUI_STATE_EVENT_HANDLER by providing the callback address of the function in a call to ***BioSPI_SubscribeToGUIEvents*** (see 9.3.3.3).

The BSP makes a callback to a framework function of this type in order to notify to the framework (and ultimately to the application) each GUI state event generated while executing a BioSPI function.

A BSP can generate GUI state events only during the execution of a call to ***BioSPI_Capture***, ***BioSPI_Verify***, ***BioSPI_Identify***, or ***BioSPI_Enroll***. A GUI state event with the moment value BioAPI_GUI_MOMENT_BEFORE_START is generated before the start of each suboperation, and a GUI state event with the moment value BioAPI_GUI_MOMENT_AFTER_END is generated after the end of each suboperation (see 7.47).

9.2.6.3 Parameters

The parameters of this function pointer type are the same as those of ***BioAPI_GUI_STATE_EVENT_HANDLER***, except for the absence of the *context address* parameter.

NOTE See also C.8.

9.2.6.4 Return Value

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

9.2.6.5 Errors

```
BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED
```

9.3 Biometric Service Provider Operations

9.3.1 SPI Component Management Operations

An application should not assume that this function is available when operating with a BSP designed for a framework-free system, as such a BSP is not required to support this function.

9.3.1.1 BioSPI_BSPLoad

```
BioAPI_RETURN BioAPI BioSPI_BSPLoad
    const BioAPI_UUID *BSPUuid,
    BioSPI_EventHandler BioAPINotifyCallback,
    BioSPI_BFP_ENUMERATION_HANDLER BFPEnumerationHandler,
    BioSPI_MEMORY_FREE_HANDLER MemoryFreeHandler);
```

9.3.1.1.1 Description

This function completes the component initialization process between BioAPI and the biometric service provider. The function ***BioSPI_BSPLoad*** shall not be called more than once without a corresponding call to ***BioSPI_BSPUnload***.

When the BioAPI version number in use is 2.1 or greater, a BSP receiving a call to this function shall immediately send one “insert” event notification for each available BioAPI Unit (of each categor) by calling back the function provided by the Framework at the address pointed to by BioAPINotifyCallback. If the biometric application has provided an event handler address in its call to BioAPI_BSPLoad, the Framework will call back, in turn, the application's event handler. If the hardware component for a given BioAPI unit is not present, the “insert” event shall not be raised until the hardware component has been plugged in.

The *BSPUuid* identifies the invoked BSP.

The *BioAPINotifyCallback* defines a callback used to notify the BioAPI Framework of events of type BioAPI_EVENT in any ongoing, attached sessions. The BSP shall retain this information for later use.

The *BFPEnumerationHandler* is the address of the BFP enumeration handler callback provided by the Framework to the BSP. The BSP shall retain this information for later use. The BSP can use the callback whenever it needs to obtain information about the BFPs installed in the Framework.

The *MemoryFreeHandler* is the address of the memory deallocation handler callback provided by the Framework to the BSP. The BSP shall retain this information for later use. The BSP shall use the callback whenever it needs to deallocate a memory block that was allocated by the Framework during a prior callback to the BFP enumeration handler.

NOTE This is a sister function to ***BioAPI_BSPLoad*** (see clause 8.1.5).

9.3.1.1.2 Parameters

— *BSPUuid (input)* – The UUID of the invoked biometric service provider, provided as a consistency check.

— *BioAPINotifyCallback (input)* – A function pointer for the BioAPI event handler that manages events of type BioAPI_EVENT.

— *BFPEnumerationHandler (input)* – A function pointer to the Framework’s BFP enumeration handler that returns BFP schema information for all the installed BFPs.

— *MemoryFreeHandler (input)* – A function pointer to the Framework's memory deallocation handler.

9.3.1.1.3 Return Value

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

9.3.1.1.4 Errors

See ***BioAPI_BSPLoad*** (clause 8.1.5).

9.3.1.2 BioSPI_BSPUnload

```
BioAPI_RETURN BioAPI BioSPI_BSPUnload
   (const BioAPI_UUID *BSPUuid);
```

9.3.1.2.1 Description

This function disables events and de-registers the event-notification function. The biometric service provider may perform cleanup operations, reversing the initialization performed in ***BioSPI_BSPLoad***.

NOTE This is a sister function to ***BioAPI_BSPUnload*** (see clause 8.1.6).

9.3.1.2.2 Parameters

— *BSPUuid (input)* – The UUID of the invoked biometric service provider.

9.3.1.2.3 Return Value

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

9.3.1.2.4 Errors

See ***BioAPI_BSPUnload*** (clause 8.1.6).

9.3.1.3 BioSPI_BSPAttach

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

9.3.1.3.1 Description

This function is invoked by the Framework once for each invocation of ***BioAPI_BSPAttach*** specifying the BSP identified by *BSPUuid*.

The biometric service provider shall verify compatibility with the version level specified by *Version*. If the version is not compatible, then this function fails. The BSP should perform all initializations required to support the new BSP invocation.

The BSP shall attach the specified BioAPI Units if they are supported.

NOTE This is a sister function to ***BioAPI_BSPAttach*** (see clause 8.1.7).

9.3.1.3.2 Parameters

— *BSPUuid (input)* – a pointer to the UUID of the invoked biometric service provider.

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

— *UnitList (input)* – a pointer to a buffer containing a list of BioAPI_UNIT_LIST_ELEMENT structures indicating to the BSP which BioAPI Units (supported by the BSP) it is to use for this attach session. The structures contain the ID and category of each BioAPI Unit. One of the following will be specified for each category of BioAPI Unit.

  a) Selection of a specific BioAPI Unit: The particular BioAPI Unit to be used in this attach session is specified by inclusion of its ID and category.

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

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

NOTE 1 Any subsequent calls requiring use of a BioAPI Unit of this category will fail with an error return.

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

— *BSPHandle (input)* – The BioAPI_HANDLE value assigned by the Framework and associated with the attach session being created by this function This value shall not be BioAPI_INVALID_BSP_HANDLE when the BioAPI version number in use is 2.1 or greater.

NOTE 2 Only one BioAPI Unit of each category can be attached for each attach session at any time.

9.3.1.3.3 Return Value

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

9.3.1.3.4 Errors

See ***BioAPI_BSPAttach*** (clause 8.1.7).

9.3.1.4 BioSPI_BSPAttachSecure (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioSPI_BSPAttachSecure
   (const BioAPI_UUID *BSPUuid,
   BioAPI_VERSION Version,
   const BioAPI_ACBio_PARAMETERS *ACBioParameters,
   const BioAPI_UNIT_LIST_ELEMENT *UnitList,
   const BioAPI_SECURITY_PROFILE *SecurityProfileList,
   uint32_t NumUnits,
   BioAPI_HANDLE BSPHandle);
```

9.3.1.4.1 Description

This function is invoked by the Framework once for each invocation of ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) specifying the BSP identified by *BSPUuid*.

The biometric service provider shall verify compatibility with the version level specified by *Version*. If the version is not compatible, then this function fails. The BSP should perform all initializations required to support the new BSP invocation.

The BSP shall attach the specified BioAPI Units if they are supported.

NOTE This is a sister function to ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater. See clause 8.1.8).

9.3.1.4.2 Parameters

— *BSPUuid (input)* – a pointer to the UUID of the invoked biometric service provider.

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

— *ACBioParameters (input/*optional*)* – Parameters about ACBio generation. A NULL pointer shall be set if no generation of ACBio instances is necessary.

— *UnitList (input)* – a pointer to a buffer containing a list of BioAPI_UNIT_LIST_ELEMENT structures indicating to the BSP which BioAPI Units (supported by the BSP) it is to use for this attach session. The structures contain the ID and category of each BioAPI Unit. One of the following will be specified for each category of BioAPI Unit.

  a) Selection of a specific BioAPI Unit: The particular BioAPI Unit to be used in this attach session is specified by inclusion of its ID and category.

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

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

  NOTE 1 Any subsequent calls requiring use of a BioAPI Unit of this category will fail with an error return.

— *SecurityProfileList (input)* – a pointer to a buffer containing a list of BioAPI_SECURITY_PROFILE structures which contain security information to be used in security operations by BioAPI Units. The order of this list shall keep that of the *UnitList*:

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

— *BSPHandle (input)* – The BioAPI_HANDLE value assigned by the Framework and associated with the attach session being created by this function.

NOTE 2 Only one BioAPI Unit of each category can be attached for each attach session at any time.

9.3.1.4.3 Return Value

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

9.3.1.4.4 Errors

See ***BioAPI_BSPAttachSecure*** (clause 8.1.8).

9.3.1.5 BioSPI_BSPDetach

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

9.3.1.5.1 Description

This function is invoked by the BioAPI Framework once for each invocation of ***BioAPI_BSPDetach*** specifying the attach session identified by *BSPHandle*. The biometric service provider shall perform all cleanup operations associated with the specified attach handle.

NOTE This is a sister function to ***BioAPI_BSPDetach*** (see clause 8.1.9).

9.3.1.5.2 Parameters

— *BSPHandle (input)* – The handle associated with the attach session being terminated by this function.

9.3.1.5.3 Return Value

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

9.3.1.5.4 Errors

See ***BioAPI_BSPDetach*** (clause 8.1.9).

9.3.1.6 BioSPI_QueryUnits

```
BioAPI_RETURN BioAPI BioSPI_QueryUnits
    (const BioAPI_UUID *Uuid,
    BioAPI_UNIT_SCHEMA **UnitSchemaArray,
    uint32_t *NumberOfElements);
```

NOTE 1 Details of the function definition are located in clause 8.1.10, ***BioAPI_QueryUnits***.

NOTE 2 The structure of BioAPI_UNIT_SCHEMA is dependent on the version of BioAPI. BioAPI_UNIT_SCHEMA of BioAPI 2.2 contains security information while that of BioAPI 2.1 or less does not.

9.3.1.7 BioSPI_QueryBFPs

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

NOTE 1 On an incoming call to this function, the BSP can use the BFP Enumeration Handler callback (see clause 9.2.2) to obtain information about all installed BFPs, and can then create the list of all supported BFPs by checking each entry of the array returned by the callback.

NOTE 2 Details of the function definition are located in clause 8.1.12, ***BioAPI_QueryBFPs***.

9.3.1.8 BioSPI_ControlUnit

```
BioAPI_RETURN BioAPI BioSPI_ControlUnit

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

NOTE Details of the function definition are located in clause 8.1.13, ***BioAPI_ControlUnit***.

9.3.1.9 BioSPI_Control (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

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

NOTE Details of the function definition are located in clause 8.1.14, ***BioAPI_Control.***

9.3.1.10 BioSPI_Transform (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioSPI_Transform
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_UUID *OperationUUID,
    const BioAPI_INPUT_BIR *InputBIRs,
    uint32_t NumberOfInputBIRs,
    BioAPI_BIR_HANDLE **OutputBIRs,
    uint32_t *NumberOfOutputBIRs);
```

NOTE Details of the function definition are located in clause 8.1.15, ***BioAPI_Transform***.

9.3.2 SPI Data Handle Operations

9.3.2.1 BioSPI_FreeBIRHandle

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

NOTE Details of the function definition are located in clause 8.2.1, ***BioAPI_FreeBIRHandle***.

9.3.2.2 BioSPI_GetBIRFromHandle

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

NOTE Details of the function definition are located in clause 8.2.2, ***BioAPI_GetBIRFromHandle***.

9.3.2.3 BioSPI_GetHeaderFromHandle

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

NOTE Details of the function definition are located in clause 8.2.3, ***BioAPI_GetHeaderFromHandle***

9.3.3 SPI Callback and Event Operations

NOTE An application should not assume that this function is available when operating with a BSP designed for a framework-free system, as such a BSP is not required to support this function.

9.3.3.1 BioSPI_EnableEvents

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

NOTE Details of the function definition are located in clause 8.3.1, ***BioAPI_EnableEvents***.

9.3.3.2 BioSPI_SetGUICallbacks (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

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

NOTE Details of the function definition are located in clause 8.3.2, ***BioAPI_SetGUICallbacks***.

9.3.3.3 **BioSPI_SubscribeToGUIEvents** (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioSPI_SubscribeToGUIEvents
   (const BioAPI_UUID *BSPUuid,
   BioSPI_GUI_SELECT_EVENT_HANDLER FwGUISelectEventHandler,
   BioSPI_GUI_STATE_EVENT_HANDLER FwGUIStateEventHandler,
   BioSPI_GUI_PROGRESS_EVENT_HANDLER FwGUIProgressEventHandler);
```

This function provides to the BSP the callback addresses of the framework's GUI select event handler, GUI state event handler, and GUI progress event handler. The three addresses shall not be NULL.

After a call to this function and until a subsequent call to ***BioSPI_UnsubscribeFromGUIEvents*** or ***BioSPI_BSPUnload***, for each GUI event that the BSP generates while executing BioSPI function calls, the BSP shall notify that event to the framework by calling back the framework GUI event handler corresponding to the type of GUI event. In addition, any BSP-controlled GUI shall be disabled during that time.

The parameter BSPUuid is provided only as a reliability measure. The BSP shall ensure that the value of this parameter matches its BSP product UUID.

Unlike the function ***BioAPI_SubscribeToGUIEvents***, this function shall not create a new subscription on each call. On each call, the BSP shall simply replace the old callback addresses (initially NULL) with the ones provided in the call.

A single call to ***BioAPI_UnsubscribeFromGUIEvents*** will clear the three callback addresses. Only one such call is needed even if the function ***BioAPI_SubscribeToGUIEvents*** has been called multiple times.

NOTE Once the framework has called this function, it will never call it again until after a call to ***BioSPI_UnsubscribeFromGUIEvents*** or ***BioSPI_BSPUnload***. The framework will pass the same three callback addresses every time it calls this function.

9.3.3.3.1 Parameters

— *BSPUuid (input)* – A UUID identifying the BSP.

— *FwGUISelectEventHandler (input)* – The callback address of a framework function that is to receive GUI select event notifications from the BSP.

— *FwGUIStateEventHandler (input)* – The callback address of a framework function that is to receive GUI state event notifications from the BSP.

— *FwGUIProgressEventHandler (input, optional)* – The callback address of a framework function that is to receive GUI progress event notifications from the BSP.

9.3.3.4 BioSPI_UnsubscribeFromGUIEvents (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioSPI_UnsubscribeFromGUIEvents
   (const BioAPI_UUID *BSPUuid);
```


This function clears the callback addresses of the framework's GUI select event handler, GUI state event handler, and GUI progress event handler in the BSP.

After a call to this function, the BSP shall cease to notify GUI events to the framework.

The parameter *BSPUuid* is provided only as a reliability measure. The BSP shall ensure that the value of this parameter matches its BSP product UUID.

9.3.3.4.1 Parameters

— *BSPUuid (input)* – A UUID identifying the BSP.

NOTE See also C.8.

9.3.4 SPI Biometric Operations

9.3.4.1 BioSPI_Capture

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

NOTE Details of the function definition are located in clause 8.4.1, ***BioAPI_Capture***.

9.3.4.2 BioSPI_CreateTemplate

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

NOTE Details of the function definition are located in clause 8.4.2, ***BioAPI_CreateTemplate***.

9.3.4.3 BioSPI_Process

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

NOTE Details of the function definition are located in clause 8.4.3, ***BioAPI_Process***.

9.3.4.4 BioSPI_ProcessWithAuxBIR

```
BioSPI_RETURN BioAPI BioSPI_ProcessWithAuxBIR
   (BioAPI_HANDLE BSPHandle,
   const BioAPI_INPUT_BIR *CapturedBIR,
   const BioAPI_INPUT_BIR *AuxiliaryData,
```

```
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
    BioAPI_BIR_HANDLE *ProcessedBIR);
```

NOTE Details of the function definition are located in clause 8.4.4, ***BioAPI_ProcessWithAuxBIR***.

9.3.4.5 BioSPI_ProcessUsingAuxBIRs (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioSPI_ProcessUsingAuxBIRs
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_INPUT_BIR *CapturedBIR,
    uint32_t NumberOfAuxBIRs,
    const BioAPI_INPUT_BIR *AuxBIRs,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
    BioAPI_BIR_HANDLE *ProcessedBIR);
```

NOTE Details of the function definition are located in clause 8.4.5, ***BioAPI_ProcessUsingAuxBIRs*** (BioAPI 2.2 or greater).

9.3.4.6 BioSPI_VerifyMatch

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

NOTE Details of the function definition are located in clause 8.4.6, ***BioAPI_VerifyMatch.***

9.3.4.7 BioSPI_ VerifyMatchUsingAuxBIRs (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioSPI_VerifyMatchUsingAuxBIRs
    (BioAPI_HANDLE BSPHandle,
    BioAPI_FMR MaxFMRRequested,
    const BioAPI_INPUT_BIR *ProcessedBIR,
    const BioAPI_INPUT_BIR *ReferenceTemplate,
    uint32_t NumberOfAuxBIRs,
    const BioAPI_INPUT_BIR *AuxBIRs,
    BioAPI_BIR_HANDLE *AdaptedBIR,
    BioAPI_BOOL *Result,
    BioAPI_FMR *FMRAchieved,
    BioAPI_BIR_HANDLE *ResultBIR,
    BioAPI_DATA *Payload);
```

NOTE Details of the function definition are located in clause 8.4.7 ***BioAPI_ VerifyMatchUsingAuxBIRs*** (BioAPI 2.2 or greater).

9.3.4.8 BioSPI_IdentifyMatch

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


NOTE Details of the function definition are located in clause 8.4.8, ***BioAPI_IdentifyMatch***.

9.3.4.9 BioSPI_Decide (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioSPI_Decide
    (BioAPI_HANDLE BSPHandle,
   const BioAPI_INPUT_BIR *ScoreBIR,
   uint32_t NumberOfAuxBIRs,
   const BioAPI_INPUT_BIR *AuxBIRs,
   BioAPI_BIR_HANDLE *DecisionBIR);
```

NOTE Details of the function definition are located in clause 8.4.9, ***BioAPI_Decide*** (BioAPI 2.2 or greater).

9.3.4.10 BioSPI_Fuse (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioSPI_Fuse
    (BioAPI_HANDLE BSPHandle,
   uint32_t NumberOfSourceBIRs,
   const BioAPI_INPUT_BIR *SourceBIRs,
   uint32_t NumberOfAuxBIRs,
   const BioAPI_INPUT_BIR *AuxBIRs,
   BioAPI_BIR_HANDLE *FusionBIR);
```

NOTE Details of the function definition are located in clause 8.4.10, ***BioAPI_Fuse*** (BioAPI 2.2 or greater).

9.3.4.11 BioSPI_Enroll

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

NOTE Details of the function definition are located in clause 8.4.11, ***BioAPI_Enroll***.

9.3.4.12 BioSPI_Verify

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

NOTE Details of the function definition are located in clause 8.4.12, ***BioAPI_Verify***.

9.3.4.13 BioSPI_Identify

```
BioAPI_RETURN BioAPI BioSPI_Identify
    (BioAPI_HANDLE BSPHandle,
    BioAPI_FMR MaxFMRRequested,
    BioAPI_BIR_SUBTYPE Subtype,
    const BioAPI_IDENTIFY_POPULATION *Population,
    uint32_t TotalNumberOfTemplates,
    BioAPI_BOOL Binning,
    uint32_t MaxNumberOfResults,
    uint32_t *NumberOfResults,
    BioAPI_CANDIDATE **Candidates,
    int32_t Timeout,
    BioAPI_BIR_HANDLE *AuditData);
```

NOTE Details of the function definition are located in clause 8.4.13, ***BioAPI_Identify.***

9.3.4.14 BioSPI_Import

```
BioAPI_RETURN BioAPI BioSPI_Import
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_DATA *InputData,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *InputFormat,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_BIR_HANDLE *ConstructedBIR);
```

NOTE Details of the function definition are located in clause 8.4.14, ***BioAPI_Import.***

9.3.4.15 BioSPI_Export (BioAPI 2.2)

```
BioAPI_RETURN BioAPI BioSPI_Export
    (BioAPI_HANDLE BSPHandle,
    BioAPI_BIR_HANDLE Handle,
    const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputBDBFormat,
    const BioAPI_BIR_SECURITY_BLOCK_FORMAT *OutputSBFormat,
    BioAPI_BIR *BIR);
```

NOTE Details of the function definition are located in clause 8.4.15, ***BioAPI_Export*** (BioAPI 2.2 or greater)

9.3.4.16 BioSPI_PresetIdentifyPopulation

```
BioAPI_RETURN BioAPI BioSPI_PresetIdentifyPopulation
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_IDENTIFY_POPULATION *Population);
```

NOTE Details of the function definition are located in clause 8.4.16, ***BioAPI_PresetIdentifyPopulation.***

9.3.5 SPI Database Operations

9.3.5.1 BioSPI_DbOpen

```
BioAPI_RETURN BioAPI BioSPI_DbOpen
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_UUID *DbUuid,
    BioAPI_DB_ACCESS_TYPE AccessRequest,
    BioAPI_DB_HANDLE *DbHandle,
    BioAPI_DB_MARKER_HANDLE *MarkerHandle);
```

NOTE Details of the function definition are located in clause 8.5.1, ***BioAPI_DbOpen***.

9.3.5.2 BioSPI_DbClose

```
BioAPI_RETURN BioAPI BioSPI_DbClose
    (BioAPI_HANDLE BSPHandle,
    BioAPI_DB_HANDLE DbHandle);
```

NOTE Details of the function definition are located in clause 8.5.2, ***BioAPI_DbClose.***

9.3.5.3 BioSPI_DbCreate

```
BioAPI_RETURN BioAPI BioSPI_DbCreate
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_UUID *DbUuid,
    uint32_t NumberOfRecords,
    BioAPI_DB_ACCESS_TYPE AccessRequest,
    BioAPI_DB_HANDLE *DbHandle);
```

NOTE Details of the function definition are located in clause 8.5.3, ***BioAPI_DbCreate.***

9.3.5.4 BioSPI_DbDelete

```
BioAPI_RETURN BioAPI BioSPI_DbDelete
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_UUID *DbUuid);
```

NOTE Details of the function definition are located in clause 8.5.4, ***BioAPI_DbDelete***.

9.3.5.5 BioSPI_DbSetMarker

```
BioAPI_RETURN BioAPI BioSPI_DbSetMarker
    (BioAPI_HANDLE BSPHandle,
    BioAPI_DB_HANDLE DbHandle,
    const BioAPI_UUID *KeyValue,
    BioAPI_DB_MARKER_HANDLE MarkerHandle);
```

NOTE Details of the function definition are located in clause 8.5.5, ***BioAPI_DbSetMarker.***

9.3.5.6 BioSPI_DbFreeMarker

```
BioAPI_RETURN BioAPI BioSPI_DbFreeMarker
    (BioAPI_HANDLE BSPHandle,
    BioAPI_DB_MARKER_HANDLE MarkerHandle);
```

NOTE Details of the function definition are located in clause 8.5.6, ***BioAPI_DbFreeMarker.***

9.3.5.7 BioSPI_DbStoreBIR

```
BioAPI_RETURN BioAPI BioSPI_DbStoreBIR
    (BioAPI_HANDLE BSPHandle,
    const BioAPI_INPUT_BIR *BIRToStore,
    BioAPI_DB_HANDLE DbHandle,
    BioAPI_UUID *BirUuid);
```

NOTE Details of the function definition are located in clause 8.5.7, ***BioAPI_DbStoreBIR.***

9.3.5.8 BioSPI_DbGetBIR

```
BioAPI_RETURN BioAPI BioSPI_DbGetBIR
    (BioAPI_HANDLE BSPHandle,
    BioAPI_DB_HANDLE DbHandle,
    const BioAPI_UUID *KeyValue,
    BioAPI_BIR_HANDLE *RetrievedBIR,
    BioAPI_DB_MARKER_HANDLE *MarkerHandle);
```

NOTE Details of the function definition are located in clause 8.5.8, ***BioAPI_DbGetBIR.***

9.3.5.9 BioSPI_DbGetNextBIR

```
BioAPI_RETURN BioAPI BioSPI_DbGetNextBIR
    (BioAPI_HANDLE BSPHandle,
    BioAPI_DB_HANDLE DbHandle,
    BioAPI_DB_MARKER_HANDLE MarkerHandle,
    BioAPI_BIR_HANDLE *RetrievedBIR,
    BioAPI_UUID *BirUuid);
```

NOTE Details of the function definition are located in clause 8.5.9, ***BioAPI_DbGetNextBIR.***

9.3.5.10 BioSPI_DbDeleteBIR

```
BioAPI_RETURN BioAPI BioSPI_DbDeleteBIR
    (BioAPI_HANDLE BSPHandle,
    BioAPI_DB_HANDLE DbHandle,
    const BioAPI_UUID *KeyValue);
```

NOTE Details of the function definition are located in clause 8.5.10, ***BioAPI_DbDeleteBIR.***

9.3.6 SPI BioAPI Unit operations

An application should not assume that this function is available when operating with a BSP designed for a framework-free system, as such a BSP is not required to support this function.

9.3.6.1 BioSPI_SetPowerMode

```
BioAPI_RETURN BioAPI BioSPI_SetPowerMode
    (BioAPI_HANDLE BSPHandle,
    BioAPI_UNIT_ID UnitId,
    BioAPI_POWER_MODE PowerMode);
```

NOTE Details of the function definition are located in clause 8.6.1, ***BioAPI_SetPowerMode.***

9.3.6.2 BioSPI_SetIndicatorStatus

```
BioAPI_RETURN BioAPI BioSPI_SetIndicatorStatus
    (BioAPI_HANDLE BSPHandle,
    BioAPI_UNIT_ID UnitId,
    BioAPI_INDICATOR_STATUS IndicatorStatus);
```

NOTE Details of the function definition are located in clause 8.6.2, ***BioAPI_SetIndicatorStatus.***

9.3.6.3 BioSPI_GetIndicatorStatus

```
BioAPI_RETURN BioAPI BioSPI_GetIndicatorStatus
    (BioAPI_HANDLE BSPHandle,
```



```
    BioAPI_UNIT_ID UnitId,
    BioAPI_INDICATOR_STATUS *IndicatorStatus);
```

NOTE Details of the function definition are located in clause 8.6.3, ***BioAPI_GetIndicatorStatus.***

9.3.6.4 BioSPI_CalibrateSensor

```
BioAPI_RETURN BioAPI BioSPI_CalibrateSensor
    (BioAPI_HANDLE BSPHandle,
    int32_t Timeout);
```

NOTE Details of the function definition are located in clause 8.6.4, ***BioAPI_CalibrateSensor.***

9.3.7 SPI Utility Functions

9.3.7.1 BioSPI_Cancel

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

NOTE Details of the function definition are located in clause 8.7.1, ***BioAPI_Cancel.***

9.3.7.2 BioSPI_Free

```
BioAPI_RETURN BioAPI BioSPI_Free
    (void* Ptr);
```

NOTE Details of the function definition are located in clause 8.7.2, ***BioAPI_Free***.

10 Component registry interface

NOTE 1 This clause does not apply for a framework-free system.

The component registry provides a registry for static information describing the capabilities of BioAPI components. The component registry may be queried by biometric applications to obtain this information. The component registry is part of the BioAPI Framework.

An application (such as an installation wizard) that installs a BioAPI component (BioAPI Framework, BSP, or BFP) posts information about that component into the component registry. This information is used by a biometric application to determine what BSPs and BFPs have been installed. The biometric application can use this information dynamically in its decision logic. For example, it can decide whether or not to make a specific (optional) function call to a given BSP depending on whether the registry entry for that BSP indicates that the call is supported. Additionally, the component registry provides information regarding the BDB formats supported by a BSP (see clause 6.7 and Annex B) and default values of a BSP for various common parameters (such as timeouts).

NOTE 2 Information regarding BioAPI Units is not stored in the component registry but is obtained upon loading a BSP (as an insert event) or in response to a query (***BioAPI_QueryUnits***).

The component registry is designed to be platform independent and does not necessarily (but could) use the built-in registry of any specific operating system (such as the Microsoft Windows™ registry).

The means of posting information to the component registry is defined in clause 10.2.

10.1 BioAPI Registry Schema

The various components of a BioAPI-based biometric system are represented by records in the component registry. These records are operating system independent. The following sub-clauses define the information that is held in the registry for each component of a BioAPI-based biometric system.

NOTE All string types defined as elements of the component schemas are character encoded as specified by ISO/IEC 10646 (UTF-8 transformation) unless otherwise specified in the data structures of clause 7.

10.1.1 Framework Schema

This schema contains attributes of the BioAPI Framework itself.

Table 2 — Framework Schema Elements

Field Name	Field Data Type	Description
FrameworkUUID	UINT8_T [16]	UUID uniquely identifying the BioAPI Framework.
Description	STRING	The BioAPI Framework product description, in text.
Path	STRING	Path (including filename) of the executable code of the BioAPI Framework
Spec Version	UINT8_T	The BioAPI Specification Version that is supported
Product Version	STRING	The BioAPI Framework product version
Vendor	STRING	The BioAPI Framework vendor name, in text.
FW Property ID	UINT8_T [16]	UUID identifying the format of the Framework Properties element (assigned by the vendor)
FW Property	DATA	Vendor specified information about the Framework

NOTE See also clause 7.40.

10.1.2 BSP Schema

This schema describes the capabilities of a BSP. There is one BSP Schema for each BSP that has been installed on the biometric system.

Table 3 — BSP Schema Elements

Field Name	Field Data Type	Description
BSPUUID	UINT8_T [16]	UUID uniquely identifying the BSP. This field shall be present only when the BioAPI version number in use is 2.0.
BSPProductUUID	UINT8_T [16]	UUID uniquely identifying the BSP as a software product. This field shall be present only when the BioAPI version number in use is 2.1 or greater.
Description	STRING	Text descriptive name of the BSP
Path	STRING	Path where the BSP executable is located, including the filename for the executable code of the BSP (may be a URL)
Spec Version	UINT8_T	The BioAPI Specification version that is supported
Product Version	STRING	The BSP product version
Vendor	STRING	The BSP vendor name, in text
[1] Applies only when a BSP is only capable of directly managing a single archive unit. Value=0 means it is capable of managing multiple units (either directly or through a BFP interface) & information about these units will be provided as part of the insert notification (part of Unit Schema).		

Table 3 *(continued)*

Field Name	Field Data Type	Description
BSP Supported Formats	MULTIUINT32_T	Multiple 4-byte Format Owner/Format Type values (BioAPI_BIR_BIOMETRIC_DATA_FORMAT). Each 32-bit value in the array specifying a supported BDB format. (The number of such 4-byte values is specified in "Number of Supported Formats" element.) See BioAPI_BIR_BIOMETRIC_DATA_FORMAT
Number of Supported Formats	UINT32_T	Integer specifying the number of BDB formats supported by the BSP (Number of 2-byte integer pairs in BSP Supported Formats)
Factors Mask	UINT32_T	A mask which indicates what forms of authentication are supported by the BSP (See BioAPI_BIR_BIOMETRIC_TYPE)
Operations	UINT32_T	Operations (functions) supported by the BSP (See BioAPI_ OPERATIONS_MASK)
Options	UINT32_T	Options supported by the BSP (See BioAPI_OPTIONS_MASK)
Payload Policy	UINT32_T	Threshold setting (maximum FMR value) used to determine when to release a payload (See BioAPI_FMR)
Max Payload Size	UINT32_T	Maximum size in bytes of a payload
Default Verify Timeout	INT32_T	Default timeout value (in milliseconds) used by the BSP for verify operations when no timeout is set by the biometric application
Default Identify Timeout	INT32_T	Default timeout value (in milliseconds) used by the BSP for identify operations when no timeout is set by the biometric application.
Default Capture Timeout	INT32_T	Default timeout value (in milliseconds) used by the BSP for capture operations when no timeout is set by the biometric application.
Default Enroll Timeout	INT32_T	Default timeout value (in milliseconds) used by the BSP for enroll operations when no timeout is set by the biometric application.
Default Calibrate Timeout	INT32_T	Default timeout value in milliseconds used by the BSP for sensor calibration operations when no timeout is specified by the application.
MAX BSP DB size	UINT32_T	Maximum size of a BIR database. If NULL, no BSP database exists for this BSP.[1]
Max Identify Population	UINT32_T	Largest population supported by the Identify function.[1] Unlimited = 0xFFFFFFFF.

[1] Applies only when a BSP is only capable of directly managing a single archive unit. Value=0 means it is capable of managing multiple units (either directly or through a BFP interface) & information about these units will be provided as part of the insert notification (part of Unit Schema).

Table 3 *(continued)*

Field Name	Field Data Type	Description
MaxNumEnroll-Instances	UINT32_T	The maximum number of distinct instances that a BSP can create reference templates in one enroll operation. This field shall be present only when the BioAPI version number in use is 2.1 or greater.
HostingEndpointIRI	UINT8_T	An IRI identifying the framework whose component registry contains a registration of the BSP. This field shall be present only when the BioAPI version number in use is 2.1 or greater.
BSPAccessUUID	UINT8_T [16]	A UUID, unique within the scope of an application, which the application may use to refer to the BSP as an alternative to the BSP product UUID. This field shall be present only when the BioAPI version number in use is 2.1 or greater.
[1] Applies only when a BSP is only capable of directly managing a single archive unit. Value=0 means it is capable of managing multiple units (either directly or through a BFP interface) & information about these units will be provided as part of the insert notification (part of Unit Schema).		

NOTE See also clause 7.22 (BioAPI 2.0) and 7.23 (BioAPI 2.1 or greater).

10.1.3 BFP Schema

This schema describes the capabilities of a BFP. There is one BFP Schema for each BFP that has been installed on the biometric system.

Table 4 — BFP Schema Elements

Field Name	Field Data Type	Description
BFP UUID	UINT8_T [16]	UUID uniquely identifying the BFP
BFP Category	UINT32_T	Category of the BFP identified by the BFP UUID
Description	STRING	Text descriptive name of the BFP
Path	STRING	Path where the BFP executable is located, including the filename for the executable code of the BFP (may be a URL)
Spec Version	UINT8_T	The BioAPI FPI Specification version supported[2]
Product Version	STRING	The BFP product version
Vendor	STRING	The BFP vendor name, in text
BFP Supported Formats	MULTIUINT32_T	Multiple 4-byte Format Owner/Format Type values (BioAPI_BIR_BIOMETRIC_DATA_FORMAT). Each 32-bit value in the array specifying a supported BDB format. (The number of such 4-byte values is specified in "Number of Supported Formats" element.) See BioAPI_BIR_BIOMETRIC_DATA_FORMAT
Number of Supported Formats	UINT32_T	Integer specifying the number of BDB formats supported by the BFP (Number of 2-byte integer pairs in BFP Supported Formats)
[2] Function Provider Interface (FPI) specifications will be published as separate parts of this IS.		

Table 4 *(continued)*

Field Name	Field Data Type	Description
Factors Mask	UINT32_T	A mask which indicates what forms of authentication are supported by the BFP (See BioAPI_BIR_BIOMETRIC_TYPE)
BFP Property ID	UINT8_T [16]	UUID identifying the format of the BFP Properties element (assigned by the vendor)
BFP Property	DATA	Vendor specified information about the BFP
[2] Function Provider Interface (FPI) specifications will be published as separate parts of this IS.		

NOTE See also clause 7.6.

10.2 Component registry functions

The following component registry utility functions are provided to ease the development of BioAPI compatible applications and BSPs.

Installation functions are provided so that a BSP or BFP implementer can populate the BSP or BFP schema within the component registry. The component registry is independent of the operating system.

10.2.1 BioAPI_Util_InstallBSP

10.2.1.1 Description

This function installs, updates (refreshes), or removes the references to a BSP in the component registry.

```
BioAPI_RETURN BioAPI_Util_InstallBSP (
   BioAPI_INSTALL_ACTION Action,
   BioAPI_INSTALL_ERROR *Error,
   const BioAPI_BSP_SCHEMA *BSPSchema);
```

Upon initiation of this function for an install, the BioAPI Framework shall create the BSP schema entry in the component registry and populate it with the content of the input BSP schema.

Upon initiation of this function for a refresh, the BioAPI Framework shall replace the existing entries for the BSP schema in the component registry with the content of the input BSP schema, based on the UUID of the BSP within that schema.

Upon initiation of this function for a deinstallation, the BioAPI Framework shall remove the BSP schema entry in the component registry for the BSP indicated by the *BSPUuid* within the input schema.

10.2.1.2 Parameters

— *Action (input)* – Installation action to be performed. Valid values are:

Value	Description
BioAPI_INSTALL_ACTION_INSTALL	Install BSP (add entry)
BioAPI_INSTALL_ACTION_REFRESH	Refresh the information in the component registry.
BioAPI_INSTALL_ACTION_UNINSTALL	Uninstall the BSP (remove entry)

— *Error (output)* – A pointer to a BioAPI_INSTALL_ERROR structure. If the function fails, the structure contains additional information regarding the error state.

— *BSPSchema (input)* – A pointer to elements of the BSP schema (defined in 7.22 (BioAPI 2.0) and 7.23 (BioAPI 2.1 or greater)) describing the features and characteristics of the BSP to be installed. When the function is called with *Action* set to BioAPI_INSTALL_ACTION_UNINSTALL, it is only necessary to set the *BSPUuid* element of the *BSPSchema* parameter.

10.2.1.3 Return Values

If the function is successful, it returns BioAPI_OK. If the function fails, it returns a valid BioAPI error code (see Clause 11). The parameter *Error* contains additional error information.

10.2.2 BioAPI_Util_InstallBFP

10.2.2.1 Description

This function installs, updates (refreshes), or removes the entries for a BioAPI BFP in the component registry.

```
BioAPI_RETURN BioAPI_Util_InstallBFP (
   BioAPI_INSTALL_ACTION Action,
   BioAPI_INSTALL_ERROR *Error,
   const BioAPI_BFP_SCHEMA *BFPSchema);
```

10.2.2.2 Parameters

— *Action (input)* – Installation action to be performed. Valid values are:

Value	Description
BioAPI_INSTALL_ACTION_INSTALL	Install BSP (add entry)
BioAPI_INSTALL_ACTION_REFRESH	Refresh the information in the component registry.
BioAPI_INSTALL_ACTION_UNINSTALL	Uninstall the BSP (remove entry)

— *Error (output)* – A pointer to a BioAPI_INSTALL_ERROR structure. If the function fails, the structure contains additional information regarding the error state.

— *BFPSchema (input/output)* – A pointer to component registry elements describing the features and characteristics of the BFP to be installed (see clause 7.6). When the function is called with *Action* set to BioAPI_INSTALL_ACTION_UNINSTALL, it is only necessary to set the *BFPUuid* element of the BFP*Schema* parameter.

10.2.2.3 Return Values

If the function is successful, it returns BioAPI_OK. If the function fails, it returns a valid BioAPI error code (see Clause 11). The parameter *Error* contains additional error information.

10.2.3 BioAPI_RegisterBSP (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_RegisterBSP
   (const uint8_t *HostingEndpointIRI,
   const BioAPI_BSP_SCHEMA *BSPSchema,
   BioAPI_BOOL Update);
```

10.2.3.1 Description

For the specification provided by this document, this function provides the same actions as as the function ***BioAPI_Util_InstallBSP*** with its *Action* parameter set to BioAPI_INSTALL_ACTION_INSTALL or BioAPI_INSTALL_ACTION_REFRESH. The value of the additional parameter *HostingEndpointIRI* shall be ignored by frameworks conforming to this document and shall be set to NULL by an application. It is provided to support interworking standards.

Unlike ***BioAPI_Util_InstallBSP***, this function does not contain an *Error* parameter. Error information is represented in the return value.

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

10.2.3.2 Parameters

— *HostingEndpointEndpointIRI* – This parameter shall be ignored by frameworks conforming to this document and shall be set to NULL by an application. It is provided to support interworking standards.

— *BSPSchema (input)* – A pointer to elements of the BSP schema (defined in 7.22 (BioAPI 2.0) and 7.23 (BioAPI 2.1 or greater)) describing the features and characteristics of the BSP to be installed.

— *Update (input)* – A boolean indicating whether the function is to update an existing BSP schema registration or to create a new registration. If the value of this parameter is zero and the BSP schema is found in the component registry, then the function shall return BioAPIERR_COMPONENT_ALREADY_REGISTERED. If the value of this parameter is nonzero and the BSP schema is not found in the component registry, the function shall return BioAPIERR_COMPONENT_NOT_REGISTERED.

10.2.3.3 Return Value

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

10.2.3.4 Errors

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

10.2.4 BioAPI_UnregisterBSP (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_UnregisterBSP
    (const uint8_t *HostingEndpointIRI,
    const BioAPI_UUID *BSPUuid);
```

10.2.4.1 Description

For the specification provided by this document, this function provides the same actions as the function ***BioAPI_Util_InstallBSP*** with its *Action* parameter set to BioAPI_INSTALL_ACTION_UNINSTALL. The value of the additional parameter *HostingEndpointIRI* shall be ignored by frameworks conforming to this document and shall be set to NULL by an application. It is provided to support interworking standards.

Unlike ***BioAPI_Util_UninstallBSP***, this function does not contain an *Error* parameter. Any error information is represented in the return value.

Applications conforming to this document shall either not call this function, or (if they call it) they shall set the parameter *HostingEndpointIRI* to NULL. The function is provided to support interworking standards.

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

10.2.4.2 Parameters

— *HostingEndpointEndpointIRI* – This parameter shall be ignored by frameworks conforming to this document and shall be set to NULL by an application. It is provided to support interworking standards.

— *BSPUuid (input)* – A pointer to the UUID of the BSP whose BSP schema registration is to be deleted from the component registry.

10.2.4.3 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.

10.2.4.4 Errors

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

10.2.5 ioAPI_RegisterBFP (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_RegisterBFP
   (const uint8_t *HostingEndpointIRI,
   const BioAPI_BFP_SCHEMA *BFPSchema,
   BioAPI_BOOL Update);
```

10.2.5.1 Description

This function performs the same actions as the function ***BioAPI_Util_InstallBFP*** with its *Action* parameter set to BioAPI_INSTALL_ACTION_INSTALL or BioAPI_INSTALL_ACTION_REFRESH. The value of the additional parameter *HostingEndpointIRI* shall be ignored by frameworks conforming to this document and shall be set to NULL by an application. It is provided to support interworking standards.

Unlike *BioAPI_Util_InstallBFP*, this function does not contain an *Error* parameter. Any error information is represented in the return value.

Applications conforming to this document shall either not call this function, or (if they call it) they shall set the parameter *HostingEndpointIRI* to NULL. The function is provided to support interworking standards.

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

10.2.5.2 Parameters

— *HostingEndpointEndpointIRI* – This parameter shall be ignored by frameworks conforming to this document and can be set to any value by an application. It is provided to support interworking standards.

— *BFPSchema (input)* – A pointer to elements of the BFP schema (defined in 7.22 (BioAPI 2.0) and 7.23 (BioAPI 2.1 or greater)) describing the features and characteristics of the BFP to be installed.

— *Update (input)* – A boolean indicating whether the function is to update an existing BFP schema registration or to create a new registration. If the value of this parameter is zero and the BFP schema is found in the component registry, then the function shall return BioAPIERR_COMPONENT_ALREADY_REGISTERED. If the value of this parameter is nonzero and the BFP schema is not found in the component registry, the function shall return BioAPIERR_COMPONENT_NOT_REGISTERED.

10.2.5.3 Return Value

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

10.2.5.4 Errors

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

10.2.6 BioAPI_UnregisterBFP (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_UnregisterBFP
   (const uint8_t *HostingEndpointIRI,
   const BioAPI_UUID *BFPUuid);
```

10.2.6.1 Description

This function performs the same actions as the function ***BioAPI_Util_InstallBFP*** with its *Action* parameter set to BioAPI_INSTALL_ACTION_UNINSTALL. The value of the additional parameter *HostingEndpointIRI* shall be ignored by frameworks conforming to this document and can be set to any value by an application. It is provided to support interworking standards.

Unlike ***BioAPI_Util_UninstallBFP***, this function does not contain an *Error* parameter. Any error information is represented in the return value.

Applications conforming to this document shall either not call this function, or (if they call it) they shall set the parameter *HostingEndpointIRI* to NULL. The function is provided to support interworking standards.

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

10.2.6.2 Parameters

— *HostingEndpointEndpointIRI* – This parameter shall be ignored by frameworks conforming to this document and can be set to any value by an application. It is provided to support interworking standards.

— *BFPUuid (input)* – A pointer to the UUID of the BFP whose BFP schema registration is to be deleted from the component registry.

10.2.6.3 Return Value

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

10.2.6.4 Errors

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

10.2.7 BioAPI_GetLastErrorInfo (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1 or greater.

```
BioAPI_RETURN BioAPI BioAPI_GetLastErrorInfo
   (BioAPI_ERROR_INFO *ErrorInfo);
```

10.2.7.1 Description

This function sets the members of the output parameter *ErrorInfo* (see 7.35) with information about the most recent error condition that caused a BioAPI function to return an error code.

This function is mainly intended to provide diagnostic information to the application. Since the possible values of the members of the *ErrorInfo* are not standardized and may vary across different platforms and implementations, normal BioAPI applications should not rely on those values for any important processing decisions.

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

10.2.7.2 Parameters

— *ErrorInfo (output)* – A data structure of type BioAPI_ERROR_INFO containing information about the last error occurred.

10.2.7.3 Return Value

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

10.2.7.4 Errors

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

11 BioAPI error handling

NOTE 1 This clause provides specification of error handling, and is applicable for a full BioAPI system and for use of a BSP designed for a framework-free operation, but in the latter case only the BSP-related and Unit-related errors will occur.

All BioAPI functions return a value of type BioAPI_RETURN. The value BioAPI_OK indicates that the function was successful. Any other value is an error value. Allowance has been made for returning error values originated from the BioAPI Framework and also error values that originate from a BSP or from a BioAPI Unit attached to a BSP.

NOTE 2 There are additional error codes defined in ISO/IEC 29141 – Tenprint Capture Using BioAPI.

11.1 Error Values and Error Codes Scheme

Error Value refers to the entire 32-bit BioAPI_RETURN value.

Error Code refers to the low-order 24 bits of the Error Value, and identifies the actual error situation.

The 8 high-order bits of the Error Value identify the source of the error, as follows:

a) The BioAPI Framework;

b) A BSP;

c) A BioAPI Unit attached to a BSP.

11.2 Error Codes and Error Value Enumeration

11.2.1 BioAPI Error Value Constants

```
#define BIOAPI_FRAMEWORK_ERROR      (0x00000000)
#define BIOAPI_BSP_ERROR            (0x01000000)
#define BIOAPI_UNIT_ERROR           (0x02000000)
```

11.2.2 Implementation-Specific Error Codes

The error codes 0x000000 – 0x0000ff are reserved for implementation-specific use.

11.2.3 General Error Codes

```
#define BioAPIERR_INTERNAL_ERROR                    (0x000101)
```
General system error; indicates that an operating system or internal state error has occurred and the system may not be in a known state.

```
#define BioAPIERR_MEMORY_ERROR                      (0x000102)
```
A memory error occurred.

```
#define BioAPIERR_INVALID_POINTER                   (0x000103)
```
An input/output function parameter or input/output field inside of a data structure is an invalid pointer.

```
#define BioAPIERR_INVALID_INPUT_POINTER             (0x000104)
```
An input function parameter or input field in a data structure is an invalid pointer.

```
#define BioAPIERR_INVALID_OUTPUT_POINTER            (0x000105)
```
An output function parameter or output field in a data structure is an invalid pointer.

```
#define BioAPIERR_FUNCTION_NOT_SUPPORTED            (0x000106)
```
The function is not implemented by the biometric service provider.

```
#define BioAPIERR_OS_ACCESS_DENIED                  (0x000107)
```
The operating system denied access to a required resource.

```
#define BioAPIERR_FUNCTION_FAILED                   (0x000108)
```
The function failed for an unknown reason.

```
#define BioAPIERR_INVALID_UUID                      (0x000109)
```
An input UUID is invalid.

(May occur if a component requested by this UUID is not present on the system or cannot be found.)

```
#define BioAPIERR_INCOMPATIBLE_VERSION              (0x00010a)
```
Version incompatibility.

(May occur if the called component cannot support the version of the BioAPI specification expected by the application.)

```
#define BioAPIERR_INVALID_DATA                      (0x00010b)
```
The data in an input parameter is invalid.

```
#define BioAPIERR_UNABLE_TO_CAPTURE                 (0x00010c)
```
The associated BSP is unable to capture raw samples from the requested BioAPI Unit.

```
#define BioAPIERR_TOO_MANY_HANDLES                  (0x00010d)
```
The associated BSP has no more space to allocate BIR handles.

```
#define BioAPIERR_TIMEOUT_EXPIRED                   (0x00010e)
```
The function has been terminated because the timeout value has expired.

```
#define BioAPIERR_INVALID_BIR                       (0x00010f)
```
The input BIR is invalid for the purpose required.

```
#define BioAPIERR_BIR_SIGNATURE_FAILURE             (0x000110)
```
The associated BSP could not validate the signature on the BIR.

```
#define BioAPIERR_UNABLE_TO_STORE_PAYLOAD           (0x000111)
```
The associated BSP is unable to store the payload.

```
#define BioAPIERR_NO_INPUT_BIRS                     (0x000112)
```
The identify population is NULL.

```
#define BioAPIERR_UNSUPPORTED_FORMAT                (0x000113)
```
The associated BSP does not support the BDB format requested.

```
#define BioAPIERR_UNABLE_TO_IMPORT                  (0x000114)
```
The associated BSP was unable to construct a BIR from the input data.

```
#define BioAPIERR_INCONSISTENT_PURPOSE                    (0x000115)
```
The purpose recorded in the BIR and/or the requested purpose are inconsistent with the function being performed.

```
#define BioAPIERR_BIR_NOT_FULLY_PROCESSED                 (0x000116)
```
The function requires a fully processed BIR.

```
#define BioAPIERR_PURPOSE_NOT_SUPPORTED                   (0x000117)
```
The BSP does not support the requested purpose.

```
#define BioAPIERR_USER_CANCELLED                          (0x000118)
```
User cancelled operation before completion or timeout.

```
#define BioAPIERR_UNIT_IN_USE                             (0x000119)
```
BSP (or BioAPI Unit attached to BSP) is currently being used by another biometric application.

```
#define BioAPIERR_INVALID_BSP_HANDLE                      (0x00011a)
```
The given BSP handle is not valid.

```
#define BioAPIERR_FRAMEWORK_NOT_INITIALIZED               (0x00011b)
```
A function has been called without initializing the BioAPI Framework.

```
#define BioAPIERR_INVALID_BIR_HANDLE                      (0x00011c)
```
BIR handle is invalid (does not exist or has been released).

```
#define BioAPIERR_CALIBRATION_NOT_SUCCESSFUL              (0x00011d)
```
The attempted calibration of a sensor unit was not able to be successfully completed.

```
#define BioAPIERR_PRESET_BIR_DOES_NOT_EXIST               (0x00011e)
```
No preset BIR population has been established.

```
#define BioAPIERR_BIR_DECRYPTION_FAILURE                  (0x00011f)
```
The BSP could not decrypt an input BIR (and thus was unable to use it for the requested operation).

```
#define BioAPIERR_IDENTIFY_IN_PROGRESS                    (0x000120)
```
Identification is in progress. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_LOW_QUALITY_REFERENCE_TEMPLATE          (0x000121)
```
Quality of reference template is low. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_NO_GUI_EVENT_HANDLER                    (0x000122)
```
GUI event handler is not set. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_TRANSFORMATION_NOT_SUPPORTED            (0x000121)
```
The BSP does not support transformation of a BIR. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_INVALID_ATTACH_SESSION                  (0x000122)
```
The attached session is invalid. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_COMPONENT_ALREADY_REGISTERED            (0x000123)
```
The schema of the BSP or BFP is already present in the component registry. This definition applies only when the version of BioAPI in use is 2.1 or greater.

```
#define BioAPIERR_COMPONENT_NOT_REGISTERED                (0x000124)
```
The schema of the BSP or BFP is not present in the component registry. This definition applies only when the version of BioAPI in use is 2.1 or greater.

11.2.4 Component Management Error Codes

```
#define BioAPIERR_COMPONENT_FILE_REF_NOT_FOUND       (0x000201)
```
A reference to the file for the component being loaded cannot be found.

```
#define BioAPIERR_BSP_LOAD_FAIL                    (0x000202)
```
Framework was unable to successfully load the BSP.

```
#define BioAPIERR_BSP_NOT_LOADED                   (0x000203)
```
BSP for which an action was requested is not loaded.

```
#define BioAPIERR_UNIT_NOT_INSERTED                (0x000204)
```
BioAPI Unit for which an action was requested is not in the inserted state.

```
#define BioAPIERR_INVALID_UNIT_ID                  (0x000205)
```
An invalid BioAPI Unit ID was requested.

```
#define BioAPIERR_INVALID_CATEGORY                 (0x000206)
```
An invalid category of BFP or BioAPI Unit was requested.

11.2.5 Database Error Values

```
#define BioAPIERR_INVALID_DB_HANDLE                (0x000300)
```
Invalid database handle.

```
#define BioAPIERR_UNABLE_TO_OPEN_DATABASE          (0x000301)
```
The associated BSP is unable to open the specified database.

```
#define BioAPIERR_DATABASE_IS_LOCKED               (0x000302)
```
The database cannot be opened for the access requested because it is locked.

```
#define BioAPIERR_DATABASE_DOES_NOT_EXIST          (0x000303)
```
The specified database does not exist.

```
#define BioAPIERR_DATABASE_ALREADY_EXISTS          (0x000304)
```
Create failed because the database already exists.

```
#define BioAPIERR_INVALID_DATABASE_NAME            (0x000305)
```
Invalid database name (UUID).

```
#define BioAPIERR_RECORD_NOT_FOUND                 (0x000306)
```
No record exists with the requested key.

```
#define BioAPIERR_MARKER_HANDLE_IS_INVALID         (0x000307)
```
The specified marker handle is invalid.

```
#define BioAPIERR_DATABASE_IS_OPEN                 (0x000308)
```
The database is already open.

```
#define BioAPIERR_INVALID_ACCESS_REQUEST           (0x000309)
```
Unrecognized access type.

```
#define BioAPIERR_END_OF_DATABASE                  (0x00030a)
```
End of database has been reached.

```
#define BioAPIERR_UNABLE_TO_CREATE_DATABASE        (0x00030b)
```
The associated BSP cannot create the database.

```
#define BioAPIERR_UNABLE_TO_CLOSE_DATABASE         (0x00030c)
```
The associated BSP cannot close the database.

```
#define BioAPIERR_UNABLE_TO_DELETE_DATABASE        (0x00030d)
```
The associated BSP cannot delete the database.

```
#define BioAPIERR_DATABASE_IS_CORRUPT              (0x00030e)
```
The specified database is corrupt.

11.2.6 Location Error Values

In various biometric technologies like fingerprint, face, iris, retina or others, biometric devices (sensor units) have three dimensions from the user's viewpoint. The error codes in this clause describe the

errors that can be caused by improper placement of a user's biometric feature on the biometric device when attempting to capture their biometric sample. Figure 4 below shows an example of a fingerprint sensor, which can be mounted vertically (Case a) or horizontally (Case b). When the placement of the biometric feature is improper, error codes are required to provide appropriate user feedback. If the sensor is placed vertically, the words 'forward' and 'backward' will be the expressions of intensity of the contact or distance, and if placed horizontally, these words will be the expressions of lengthwise distances.

If the BSP or BioAPI Unit generating the error code knows whether the biometric device is oriented vertically or horizontally, it can then choose an error code that describes the user's situation appropriately. If it does not, then a more general error code has to be used.

NOTE These errors can be returned from **BioAPI_CreateTemplate**, **BioAPI_Process**, **BioAPI_ProcessWithAuxBIR**, **BioAPI_VerifyMatch**, and **BioAPI_IdentifyMatch**. The BSP controlled GUI in the other biometric operations (**BioAPI_Capture**, **BioAPI_Enroll**, **BioAPI_Verify**, and **BioAPI_Identify**) will instruct the user to capture a correct sample.

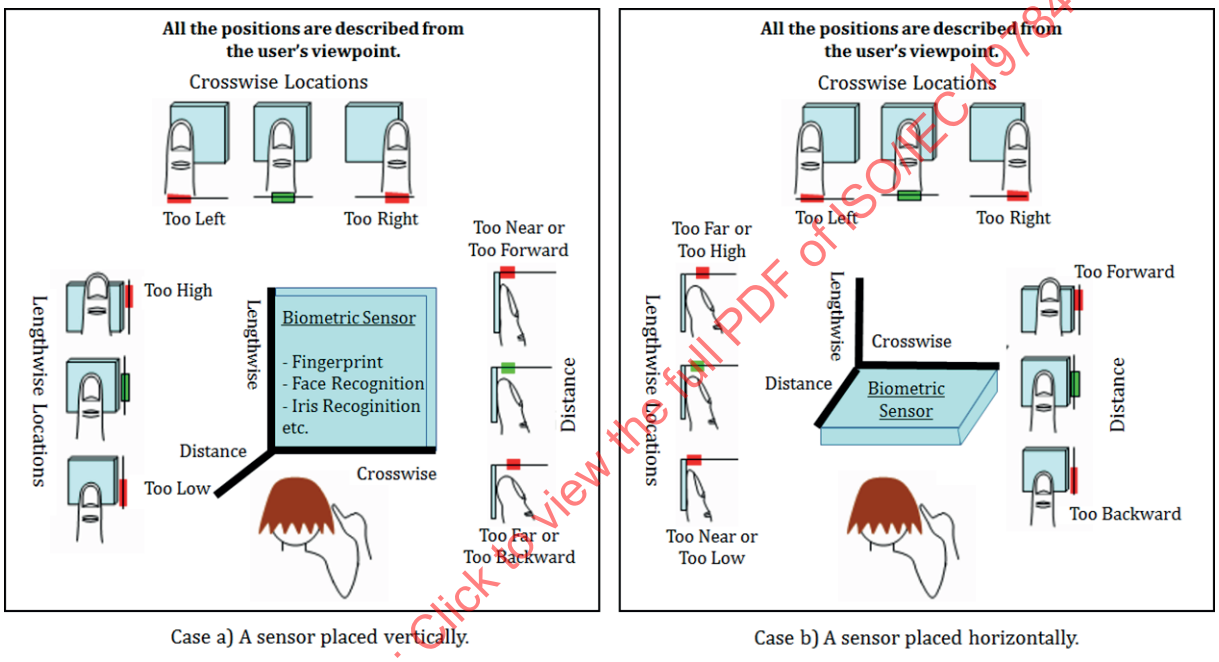

Figure 4 — Biometric Sample Locations (Example of a fingerprint sensor)

Biometric applications can use these error codes for user feedback to obtain better samples regardless of what kind of biometric technology they are using, as long as it has three dimensions. The error codes are specified below.

If the BSP or BioAPI Unit generating the error code is not certain about the orientation of the sensor, it can return generic error codes.

11.2.6.1 General location error codes

```
#define BioAPIERR_LOCATION_ERROR                    (0x000400)
```
A general location error.

```
#define BioAPIERR_OUT_OF_FRAME                      (0x000401)
```
Invalid horizontal or vertical position.

```
#define BioAPIERR_INVALID_CROSSWISE_POSITION        (0x000402)
```
Invalid crosswise position.

```
#define BioAPIERR_INVALID_LENGTHWISE_POSITION       (0x000403)
```
Invalid lengthwise position.

```
#define BioAPIERR_INVALID_DISTANCE              (0x000404)
```
Invalid distance.

11.2.6.2 Specific location error codes

```
#define BioAPIERR_LOCATION_TOO_RIGHT            (0x000405)
```
The position was too far to the right.

```
#define BioAPIERR_LOCATION_TOO_LEFT             (0x000406)
```
The position was too far to the left.

```
#define BioAPIERR_LOCATION_TOO_HIGH             (0x000407)
```
The position was too high.

```
#define BioAPIERR_LOCATION_TOO_LOW              (0x000408)
```
The position was too low.

```
#define BioAPIERR_LOCATION_TOO_FAR              (0x000409)
```
The position was too far away.

```
#define BioAPIERR_LOCATION_TOO_NEAR             (0x00040a)
```
The position was too near (close).

```
#define BioAPIERR_LOCATION_TOO_FORWARD          (0x00040b)
```
The position was too far forward.

```
#define BioAPIERR_LOCATION_TOO_BACKWARD         (0x00040c)
```
The position was too far backward.

11.2.7 Quality Error Codes

```
#define BioAPIERR_QUALITY_ERROR                 (0x000501)
```
Sample quality is too poor for the operation to succeed.

NOTE Quality errors can be returned from any function which receives a BioAPI BIR input.

```
#define BioAPIERR_TOO_EARLY                     (0x000502)
```
The behavior of the subject was too early. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_TOO_LATE                      (0x000503)
```
The behavior of the subject was too late. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_TOO_FAST                      (0x000504)
```
The behavior of the subject was too fast. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_TOO_SLOW                      (0x000505)
```
The behavior of the subject was too slow. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_TOO_LONG                      (0x000506)
```
The captured data was too long. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_TOO_SHORT                     (0x000507)
```
The captured data was too short. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_TOO_LARGE                     (0x000508)
```
The captured data was too large. This definition applies only when the BioAPI version number in use is 2.1 or greater.

```
#define BioAPIERR_TOO_SMALL                     (0x000509)
```

The captured data was too small. This definition applies only when the BioAPI version number in use is 2.1 or greater.

11.2.8 Security Error Codes (BioAPI 2.2)

```
#define  BioAPIERR_SECURITY_PROFILE_NOT_SET                (0x000601)
```
The parameter of the type BioAPI_SECURITY_PROFILE is NULL when ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) is called.

```
#define  BioAPIERR_SECURITY_ENCRYPTION_ALG_NOT_SUPPORTED   (0x000610)
```
The specified encryption algorithm is not supported.

```
#define  BioAPIERR_SECURITY_ENCRYPTION_ALG_NOT_SET         (0x000611)
```
The encryption algorithm is not set though encryption is specified.

```
#define  BioAPIERR_SECURITY_ENCKEY_NOT_SET                 (0x000612)
```
The encryption key is not set though the encryption is specified.

```
#define  BioAPIERR_SECURITY_MAC_ALG_NOT_SUPPORTED          (0x000620)
```
The specified MAC algorithm is not supported.

```
#define  BioAPIERR_SECURITY_MAC_ALG_NOT_SET                (0x000621)
```
The MAC key is not set though the MAC is specified.

```
#define  BioAPIERR_SECURITY_MACKEY_NOT_SET                 (0x000622)
```
The MAC algorithm is not set though MAC is specified.

```
#define  BioAPIERR_SECURITY_DIGITAL_SIGNATURE_ALG_NOT_SUPPORTED  (0x000640)
```
The specified digital signature algorithm is not supported.

```
#define  BioAPIERR_SECURITY_DIGITAL_SIGNATURE_ALG_NOT_SET  (0x000641)
```
The digital signature algorithm is not set though digital signature is specified.

```
#define  BioAPIERR_SECURITY_CHALLENGE_NOT_SET              (0x000701)
```
The parameter Challenge is NULL or * Challenge is not set though ACBio instance generation is requested when ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) is called.

```
#define  BioAPIERR_SECURITY_BPUIOINDEX_NOT_SET             (0x000702)
```
The parameter InitialBPUIOIndexOutput is NULL though ACBio instance generation is requested when ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) is called.

```
#define  BioAPIERR_SECURITY_SUPREMUM_BPUIOINDEX_NOT_SET    (0x000704)
```
The parameter SupremumBPUIOIndexOutput is NULL though ACBio instance generation is requested when ***BioAPI_BSPAttachSecure*** (BioAPI 2.2 or greater) is called.

```
#define  BioAPIERR_SECURITY_MAC_ALG_ACBIO_NOT_SUPPORTED    (0x000720)
```
The specified MAC algorithm for ACBio generation is not supported.

```
#define  BioAPIERR_SECURITY_MAC_ALG_ACBIO_NOT_SET          (0x000721)
```
The MAC algorithm for ACBio generation is not set though MAC for ACBio generation is specified.

```
#define  BioAPIERR_SECURITY_MACKEY_ACBIO_NOT_SET           (0x000722)
```
The MAC key for ACBio generation is not set though the MAC for ACBio generation is specified.

```
#define  BioAPIERR_SECURITY_DIGITAL_SIGNATURE_ALG_ACBIO_NOT_SUPPORTED   (0x000740)
```
The specified digital signature algorithm for ACBio generation is not supported.

```
#define  BioAPIERR_SECURITY_DIGITAL_SIGNATURE_ALG_ACBIO_NOT_SET   (0x000741)
```
The digital signature algorithm for ACBio generation is not set though digital signature for ACBio generation is specified.

```
#define  BioAPIERR_SECURITY_HASH_ALG_ACBIO_NOT_SUPPORTED   (0x000780)
```
The specified hash algorithm is not supported.

```
#define  BioAPIERR_SECURITY_HASH_ALG_ACBIO_NOT_SET         (0x000781)
```
The hash algorithm is not set though ACBio generation is specified.


```
#define  BioAPIERR_SECURITY_ENCRYPTION_FAILURE              (0x000614)
```
The encryption is failure.

```
#define  BioAPIERR_SECURITY_DECRYPTION_FAILURE              (0x000618)
```
The decryption is failure.

```
#define  BioAPIERR_SECURITY_MAC_GENERATION_FAILURE          (0x000624)
```
The MAC generation is failure.

```
#define  BioAPIERR_SECURITY_MAC_VERIFICATION_FAILURE        (0x000628)
```
The MAC verification is failure.

```
#define  BioAPIERR_SECURITY_DIGITAL_SIGNATURE_GENERATION_FAILURE  (0x000644)
```
The digital signature generation is failure.

```
#define  BioAPIERR_SECURITY_DIGITAL_SIGNATURE_VERIFICATION_FAILURE  (0x000648)
```
The digital signature verification is failure.

Annex A
(normative)

Conformance

A.1 General

A.1.1 Conformance to this document falls into the following five classes:

— BioAPI conformant biometric application;

— BioAPI 2.0 conformant BioAPI framework;

— BioAPI 2.1 conformant BioAPI framework;

— BioAPI 2.2 conformant BioAPI framework;

— BioAPI 2.0 conformant BSP, comprising one of the following subclasses:

  — BioAPI 2.0 conformant Verification BSP;

  — BioAPI 2.0 conformant Identification BSP;

  — BioAPI 2.0 conformant Capture BSP;

  — BioAPI 2.0 conformant Verification Engine;

  — BioAPI 2.0 conformant Identification Engine;

— BioAPI 2.1 conformant BSP, comprising one of the following subclasses:

  — BioAPI 2.1 conformant Verification BSP;

  — BioAPI 2.1 conformant Identification BSP;

  — BioAPI 2.1 conformant Capture BSP;

  — BioAPI 2.1 conformant Verification Engine;

  — BioAPI 2.1 conformant Identification Engine;

— BioAPI 2.2 conformant BSP, comprising one of the following subclasses:

  — BioAPI 2.2 conformant Verification BSP;

  — BioAPI 2.2 conformant Identification BSP;

  — BioAPI 2.2 conformant Capture BSP;

  — BioAPI 2.2 conformant Verification Engine;

  — BioAPI 2.2 conformant Identification Engine.

A.1.2 Conformance requirements for biometric applications, BioAPI Frameworks, and for BSPs are defined in clauses A.2, A.3, and A.4, respectively.

NOTE Conformance of BFPs is not addressed in this document, but is specified in subsequent parts of the ISO/IEC 19784 series. Interfaces to BioAPI Units are not standardized.

A.2 BioAPI Conformant Biometric Application

A.2.1 To claim compliance to the BioAPI specification, a biometric application shall, for each BioAPI function call utilized, invoke that operation consistently with this specification. That is, all input parameters shall be present and valid. The application shall accept all valid output parameters and return values.

A.2.2 There is no minimum set of functions that are required to be called. However, the biometric application shall conform to the call dependencies identified for the functions.

A.3 BioAPI Conformant Framework

NOTE This sub-clause is not applicable in a framework-free system.

A.3.1 The BioAPI Framework component is the middleware infrastructure between BioAPI conformant applications and BSPs. It serves the following general purposes:

a) BSP loading/attaching;

b) BSP and BFP management;

c) component registry maintenance and management;

d) function call pass-through/API-SPI translation;

e) handling of event notifications from BSPs, and sending those event notifications to (possibly multiple) event handlers in applications that have loaded that BSP;

f) supporting API calls related to the installation or de-installation of BioAPI components, with appropriate update of the component registry;

g) supporting queries from a BSP about installed BFPs.

A.3.2 To claim conformance to the BioAPI specification (version 2.0, version 2.1, or version 2.2), a BioAPI Framework shall:

a) provide component management functions as specified in clause 8.1;

b) provide component registry services in accordance with Clause 10;

c) perform translation of API functions specified in Clause 8 into the corresponding SPI functions specified in Clause 9;

d) conform to the data structures as defined in Clause 7 and the error codes as defined in Clause 11 when implementing a) through c), above;

e) handle event notifications and callbacks as defined in clauses 7.38, 8.3, and 9.2.

A.3.3 A conformant BioAPI 2.0 Framework is required to support all the mandatory and optional provisions of this document, except the subclauses and definitions that are specified as applying to version 2.1 or greater.

A.3.4 A conformant BioAPI 2.1 Framework is required to support all the mandatory and optional provisions of this document, except the subclauses and definitions that are specified as applying to version 2.0, version 2.2, or greater.

A.3.5 A conformant BioAPI 2.2 Framework is required to support all the mandatory and optional provisions of this document, except the subclauses and definitions that are specified as applying to version 2.0 or version 2.1.

A.3.6 No subsets of conformant Framework functions are defined.

A.4 BioAPI Conformant BSPs

To claim conformance to the BioAPI specification (version 2.0, version 2.1, or version 2.2), BSPs shall implement mandatory functions for their conformance sub-class, as defined below, in accordance with the SPI defined in Clause 9. BSPs claim conformance to one of the conformance sub-classes specified in clause A.1.1.

BSPs that claim conformance only for framework-free operation need not support the functions identified in earlier text as not being required for framework-free operation.

BioAPI 2.0 conformant BSPs (of any conformance subclass, including framework-free operation) are required to support all the mandatory provisions of this International Standard according to their conformance subclass, except the subclauses and definitions that are specified as applying to version 2.1 or greater.

BioAPI 2.1 conformant BSPs (of any conformance subclass, including framework-free operation) are required to support all the mandatory provisions of this International Standard according to their conformance subclass, except the subclauses and definitions that are specified as applying to version 2.0, version 2.2, or greater.

BioAPI 2.2 conformant BSPs (of any conformance subclass, including framework-free operation) are required to support all the mandatory provisions of this International Standard according to their conformance subclass, except the subclauses and definitions that are specified as applying to version 2.0 or version 2.1 only.

BSPs shall accept all valid input parameters and return valid outputs. Optional capabilities and returns are not required to claim conformance; but any optional functions or parameters that are implemented shall be implemented in accordance with the specification requirements, including functions not required for framework-free operation, but actually implemented. Additional parameters shall not be required.

The BSP installation process (not applicable for a BSP designed for framework-free operation) shall perform the population of all required component registry entries.

BSPs shall possess a valid and unique UUID that is associated with a specific BSP product and version.

NOTE The UUID may be self-generated (see ISOIEC 9834-8) and should (but need not) be the same on multiple systems where the same BSP product/version is installed.

BIRs generated by the BSP shall conform to the data structures defined in Clause 7 (they shall be BioAPI BIRs). BSPs shall only return BioAPI_BIR data containing a registered FormatOwner with an associated valid FormatType (see clause 7.9 and Annex B).

BSPs shall perform error handling as defined in Clause 11.

All BSPs claiming to support a full BioAPI system shall support basic Component Management (clause 9.3.1), Handle (clause 9.3.2), Event (9.3.3.1) and Utility (clause 9.3.7) operations (BSP's claiming only to support use in a framework-free system need not support any of these operations). Callback (clause 9.3.3.2), Database (clause 9.3.5), and BioAPI Unit (clause 9.3.6) operations are optional.

The following table is a summary of BSP conformance requirements by subclass of BSP. Details are provided in the following sub-clauses. Clause A.4.6 addressed conformance with respect to optional capabilities.

Table A.1 — BSP Conformance Sub-Classes

Function	Verification BSP	Identification BSP	Capture BSP	Verification Engine	Identification Engine
Component Management Functions (BSPs claiming only conformance for use in a framework-free system)					
BioSPI_BSPLoad	X	X	X	X	X
BioSPI_BSPUnload	X	X	X	X	X
BioSPI_BSPAttach	X	X	X	X	X
BioSPI_BSPAttachSecure (BioAPI 2.2)					
BioSPI_BSPDetach	X	X	X	X	X
BioSPI_QueryUnits	X	X	X		
BioSPI_QueryBFPs					
BioSPI_ControlUnit					
BioSPI_Control (BioAPI 2.1)					
BioSPI_Transform (BioAPI 2.1)					
Handle Functions (A BSP claiming only conformance for use in a framework-free system need not implement handle functions, provided it never returns a handle (recommended). If it returns a handle, then these functions shall be supported.)					
BioSPI_FreeBIRHandle	X	X	X	X	X
BioSPI_GetBIRFromHandle	X	X	X	X	X
BioSPI_GetHeaderFromHandle	X	X	X	X	X
Callback and Event Functions (None are required for a BSP claiming only conformance for use in a framework-free system)					
BioSPI_EnableEvents	X	X	X	X	X
BioSPI_SetGUICallbacks (BioAPI 2.0)					
BioSPI_SubscribeToGUIEvents (BioAPI 2.1)					
BioSPI_UnsubscribeFromGUIEvents (BioAPI 2.1)					
Biometric Functions					
BioSPI_Capture			X		
BioSPI_CreateTemplate				X	X
BioSPI_Process				X	X
BioSPI_ProcessWithAuxBIR					
BioSPI_ProcessUsingAuxBIRs (BioAPI 2.2)					
BioSPI_VerifyMatch				X	X

Table A.1 *(continued)*

Function	Verification BSP	Identification BSP	Capture BSP	Verification Engine	Identification Engine
BioSPI_VerifyMatchUsingAuxBIRs (BioAPI 2.2)					
BioSPI_IdentifyMatch					X
BioSPI_Decide (BioAPI 2.2)					
BioSPI_Fuse (BioAPI 2.2)					
BioSPI_Enroll	X	X			
BioSPI_Verify	X	X			
BioSPI_Identify		X			
BioSPI_Import					
BioSPI_PresetIdentifyPopulation					
Database Functions					
BioSPI_DbOpen					
BioSPI_DbClose					
BioSPI_DbCreate					
BioSPI_DbDelete					
BioSPI_DbSetMarker					
BioSPI_DbFreeMarker					
BioSPI_DbStoreBIR					
BioSPI_DbGetBIR					
BioSPI_DbGetNextBIR					
BioSPI_DbDeleteBIR					
BioAPI Unit Functions **(None are required for a BSP claiming only conformance for use in a framework-free system)**					
BioSPI_SetPowerMode					
BioSPI_SetIndicatorStatus					
BioSPI_GetIndicatorStatus					
BioSPI_CalibrateSensor					
Utility Functions **(None are required for a BSP claiming only conformance for use in a framework-free system)**					
BioSPI_Cancel	X	X	X	X	X
BioSPI_Free	X	X	X	X	X
GUI events					
GUI select events (BioAPI 2.1)					
GUI state events (BioAPI 2.1)					
GUI progress events (BioAPI 2.1)					

A.4.1 BioAPI Conformant Verification BSPs

Verification BSPs are those which are capable of performing 1:1 matching (or authentication), but not 1:N identification matching.

NOTE 1:few (one-to-few) matching may be supported as a series of 1:1 calls.

A BioAPI conformant Verification BSP shall support the following biometric functions:

— ***BioSPI_Verify***;

— ***BioSPI_Enroll***.

A.4.1.1 ***BioSPI_Verify*.** Only the nearest, better supported *MaxFMRRequested* is required to be supported; however, the BSP shall return that supported value (*FMRAchieved*). (The BSP shall indicate in the component registry whether it implements coarse scoring.)

NOTE See clause C.4 for an explanation of use of FMR for scoring and thresholding.

Acceptance and use of *Subtype* is optional.

Return of *Payload* is required only if one is associated with the input *ReferenceTemplate* and if the score sufficiently exceeds the *FMRAchieved* (the BSP shall post minimum FMR required to return payload in the component registry).

Return of *AdaptedBIR* is optional. Return of raw data (*AuditData*) is optional.

All BSPs shall provide any necessary user interface (as a default user interface) associated with the capture portion of the ***Verify*** operation. Support for application control of a GUI is optional.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.1.2 ***BioSPI_Enroll*.** The *Purpose* value BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY shall be accepted. If a different purpose value is requested but not supported, an error condition shall be set as a BioAPI_RETURN.

Acceptance of *Payload* is optional (i.e., if a payload is provided, it doesn't have to be accepted if the BSP does not support payload carry).

Acceptance and use of *Subtype* is optional.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input *OutputFormat* and return the *NewTemplate* in that format.

Use of the input *ReferenceTemplate*, when provided, to create the output *NewTemplate* is optional.

Return of raw data (*AuditData*) is optional.

All BSPs shall provide any necessary user interface (as a default user interface) associated with the capture portion of the ***Enroll*** operation. Support for application control of a GUI is optional.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.2 BioAPI Conformant Identification BSPs

Identification BSPs are those which are capable of performing both 1:N identification matching as well as 1:1 matching (or verification). A BioAPI conformant Identification BSP shall support the following biometric functions:

— ***BioSPI_Verify***;

— ***BioSPI_Identify***;

— ***BioSPI_Enroll***.

A.4.2.1 ***BioSPI_Verify***. Only the nearest, better supported *MaxFMRRequested* must be supported; however, the BSP shall return that supported value (*FMRAchieved*). (The BSP shall indicate in component registry whether it implements coarse scoring.)

NOTE See clause C.4 for an explanation of use of FMR for scoring and thresholding.

Acceptance and use of *Subtype* is optional.

Return of *Payload* is required only if one is associated with the input *ReferenceTemplate* and if the score sufficiently exceeds the *FMRAchieved* (the BSP shall post minimum FMR required to return payload in the component registry).

Return of *AdaptedBIR* is optional. Return of raw data (*AuditData*) is optional.

As a default, all BSPs shall provide any GUI associated with the capture portion of the ***Verify*** operation. However, support for application control of the GUI is optional.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.2.2 ***BioSPI_Identify*.** Only the nearest, better supported *MaxFMRRequested* must be supported. Return of matching *Candidates* is required; however, the BSP may return values for the *FMRAchieved* field as next nearest step/increment. (The BSP shall indicate in component registry whether it implements coarse scoring.)

NOTE See clause C.4 for an explanation of the use of FMR for scoring and thresholding.

Acceptance and use of *Subtype* is optional.

Support of binning is optional.

Return of raw data (*AuditData*) is optional.

As a default, all BSPs shall provide any GUI associated with the capture portion of the ***Identify*** operation. However, support for application control of the GUI is optional.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.2.3 ***BioSPI_Enroll*.** Only the *Purpose* values BioAPI_PURPOSE_ENROLL, BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY, and BioAPI_PURPOSE_ENROLL_FOR_IDENTIFICATION ONLY must be accepted. If another purpose is set but not supported, an error condition shall be set as a BioAPI_RETURN. Acceptance of *Payload* is optional.

Acceptance and use of *Subtype* is optional.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input *OutputFormat* and return the *NewTemplate* in that format.

Use of the input *ReferenceTemplate*, when provided, to create the output *NewTemplate* is optional.

Return of raw data (*AuditData*) is optional.

As a default, all BSPs shall provide any GUI associated with the capture portion of the ***Enroll*** operation. However, support for application control of the GUI is optional.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.3 BioAPI Conformant Capture BSPs

BioAPI Capture BSPs are BSPs which provide an interface to one or more biometric sensors and return intermediate BIRs that can then be used by other BSPs (such as a BioAPI Biometric Engine, see clauses A.4.4 and A.4.5). This type of BSP does not provide the capability to process or match the data it captures. A BioAPI conformant Capture BSP shall support the following biometric functions:

— ***BioSPI_Capture***.

A.4.3.1 *BioSPI_Capture*. Return of raw data (*AuditData*) is optional.

Acceptance and use of *Subtype* is optional.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input *OutputFormat* and return the *CapturedBIR* in that format.

As a default, all BSPs shall provide any GUI associated with the ***BioSPI_Capture*** operation. However, support for application control of the GUI is optional.

A.4.4 BioAPI Conformant Verification Engines

BioAPI Conformant Verification Engines are BSPs which contain the biometric processing and 1:1 matching algorithms, but do not perform the biometric capture. These are typically used in conjunction with another BSP which handle the capture operation (either a Capture or full BSP). A BioAPI compliant biometric engine shall support the following biometric functions:

— ***BioSPI_CreateTemplate***;

— ***BioSPI_Process***;

— ***BioSPI_VerifyMatch***.

A.4.4.1 *BioSPI_CreateTemplate*. Only the *Purpose* value BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY must be accepted. If another purpose is set but not supported, an error condition shall be set as a BioAPI_RETURN.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input *OutputFormat* and return the *NewTemplate* in that format.

Acceptance of *Payload* is optional. Template update (through *ReferenceTemplate* input) is optional.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.4.2 ***BioSPI_Process***. Only input *CapturedBIR*s with the *Purpose* value of BioAPI_PURPOSE_VERIFY must be accepted. If another purpose value is specified, an error condition shall be set as a BioAPI_RETURN.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input *OutputFormat* and return the *ProcessedBIR* in that format.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.4.3 ***BioSPI_VerifyMatch*.** Only input BIRs (*ProcessedBIR*) with a *Purpose* value of BioAPI_PURPOSE_VERIFY, and (*ReferenceTemplate*) with a *Purpose* value of either BioAPI_PURPOSE_ENROLL or BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY must be accepted. If another purpose is set, an error condition shall be set as a BioAPI_RETURN.

Only the nearest, better supported *MaxFMRRequested* must be supported; however, the BSP shall return that supported value (*FMRAchieved*). (The BSP shall indicate in component registry whether it implements coarse scoring.)

NOTE See clause C.4 for an explanation of the use of FMR for scoring and thresholding.

Return of *Payload* is required only if one is associated with the input *ReferenceTemplate* and if the score is better than the *FMRAchieved* (the BSP shall post minimum FMR required to return payload in the component registry).

Return of *AdaptedBIR* is optional.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.5 BioAPI Conformant Identification Engines

The requirements for a BioAPI conformant Identification Engine are identical to those of a Verification Engine, with the exception that ***BioSPI_CreateTemplate*** shall also accept a *CapturedBIR* with a *Purpose* value of BioAPI_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY and ***BioSPI_Process*** shall also accept a *CapturedBIR* with a *Purpose* value of BioAPI_PURPOSE_IDENTIFY as well as the addition of the following required function:

— ***BioSPI_IdentifyMatch***.

A.4.5.1 BioSPI_IdentifyMatch

Only the nearest, better supported *MaxFMRRequested* must be supported. Return of matching *Candidates* is required; however, the BSP may return values for the *FMRAchieved* field as next nearest step/increment. (The BSP shall indicate in component registry whether it implements coarse scoring.)

NOTE See clause C.4 for an explanation of the use of FMR for scoring and thresholding.

Support of binning is optional.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6 Optional Capabilities

The following capabilities are considered optional in terms of BSP support and compliance. Note that:

— If implemented, optional capabilities shall conform to specification definitions.

— BSPs are required to post (at installation) to the component registry (via the OptionsMask element of the *BSPSchema* within the ***BioAPI_Util_InstallBSP*** registry function, see clause 10.2.1) whether or not each option is supported.

NOTE This is not applicable for BSPs designed for framework-free systems.

— BSP documentation shall include a table that identifies which options are supported and which options are not supported.

A.4.6.1 Optional Functions

A.4.6.1.1 Primitive Functions

Although optional, it is highly recommended that, if supported by the underlying technology, the following primitive functions are supported in in all BSPs.

A.4.6.1.1.1 ***BioSPI_Capture***. If this function is supported, Verification BSPs need only accept the *Purpose* values BioAPI_PURPOSE_VERIFY or BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY. If another purpose value is specified and not supported, an error condition shall be set as a BioAPI_RETURN. This function shall return *CapturedBIR* with a purpose of BioAPI_PURPOSE_VERIFY, BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY, or BioAPI_PURPOSE_ENROLL accordingly.

If this function is supported, Identification BSPs shall accept all purpose values (except BioAPI_PURPOSE_AUDIT), even if there is no difference in the content or format of the returned data.

Acceptance and use of *Subtype* is optional.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input *OutputFormat* and return the *CapturedBIR* in that format.

Return of raw data (*AuditData*) is optional.

As a default, BSPs shall provide any GUI associated with the ***BioSPI_Capture*** operation. However, support for application control of the GUI is optional.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.1.2 ***BioSPI_CreateTemplate***. If this function is supported, Verification BSPs need only accept input *CapturedBIR*s with the *Purpose* value of BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY. If another purpose value is specified and not supported, an error condition shall be set as a BioAPI_RETURN.

If this function is supported, Identification BSPs shall accept input *CapturedBIR*s with all Enroll purpose values.

Acceptance of *Payload* is optional. Template update (through *ReferenceTemplate* input) is optional.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input *OutputFormat* and return the *NewTemplate* in that format.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.1.3 ***BioSPI_Process***. If this function is supported, Verification BSPs need only accept input *CapturedBIR*s with the *Purpose* value of BioAPI_PURPOSE_VERIFY. If another purpose value is specified and not supported, an error condition shall be set as a BioAPI_RETURN.

If this function is supported, Identification BSPs shall accept input *CapturedBIR*s with the *Purpose* value of either BioAPI_PURPOSE_VERIFY or BioAPI_PURPOSE_IDENTIFY, even if there is no difference in the content or format of the returned data.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input *OutputFormat* and return the *ProcessedBIR* in that format.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.1.4 ***BioSPI_ProcessWithAuxBIR*** (BioAPI2.0 and BioAPI2.1). If this function is supported, Verification BSPs need only accept input *CapturedBIR*s with the *Purpose* value of BioAPI_PURPOSE_VERIFY. If another purpose value is specified and not supported, an error condition shall be set as a BioAPI_RETURN.

If this function is supported, Identification BSPs shall accept input *CapturedBIR*s with the *Purpose* value of either BioAPI_PURPOSE_VERIFY or BioAPI_PURPOSE_IDENTIFY, even if there is no difference in the content or format of the returned data.

BSPs supporting this function will include in their documentation the format and content (or alternatively the source) of the *AuxiliaryData* which it accepts as input.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input *OutputFormat* and return the *ProcessedBIR* in that format.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.1.5 ***BioSPI_ProcessUsingAuxBIRs*** (BioAPI2.2). If this function is supported, Verification BSPs need only accept input CapturedBIRs with the Purpose value of BioAPI_PURPOSE_VERIFY. If another purpose value is specified and not supported, an error condition shall be set as a BioAPI_RETURN.

If this function is supported, Identification BSPs shall accept input CapturedBIRs with the Purpose value of either BioAPI_PURPOSE_VERIFY or BioAPI_PURPOSE_IDENTIFY, even if there is no difference in the content or format of the returned data.

BSPs supporting this function will include in their documentation the format and content (or alternatively the source) of the AuxiliaryData which it accepts as input.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input OutputFormat and return the ProcessedBIR in that format.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList,* corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.1.6 ***BioSPI_VerifyMatch***. If this function is supported, only input BIRs (*ProcessedBIR*) with a *Purpose* value of BioAPI_PURPOSE_VERIFY and *ReferenceTemplates* with a purpose value of either

BioAPI_PURPOSE_ENROLL or BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY must be accepted. If another purpose is set and not supported, an error condition shall be set as a BioAPI_RETURN.

Only the nearest, better supported *MaxFMRRequested* must be supported; however, the BSP shall return that supported value (*FMRAchieved*). (The BSP shall indicate in component registry whether it implements coarse scoring.)

NOTE See clause C.4 for an explanation of the use of FMR for scoring and thresholding.

Return of *Payload* is required only if one is associated with the input *ReferenceTemplate* and if the score is better than the *FMRAchieved* (the BSP shall post minimum FMR required to return payload in the component registry).

Return of *AdaptedBIR* is optional.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.1.7 ***BioSPI_VerifyMatchUsingAuxBIRs*** (BioAPI2.2). If this function is supported, only input BIRs (ProcessedBIR) with a Purpose value of BioAPI_PURPOSE_VERIFY and ReferenceTemplates with a purpose value of either BioAPI_PURPOSE_ENROLL or BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY must be accepted. If another purpose is set and not supported, an error condition shall be set as a BioAPI_RETURN.

Only the nearest, better supported MaxFMRRequested must be supported; however, the BSP shall return that supported value (FMRAchieved). (The BSP shall indicate in component registry whether it implements coarse scoring.)

NOTE See clause C.4 for an explanation of the use of FMR for scoring and thresholding.

Return of Payload is required only if one is associated with the input ReferenceTemplate and if the score is better than the FMRAchieved (the BSP shall post minimum FMR required to return payload in the component.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList,* corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.1.8 ***BioSPI_IdentifyMatch***. If this function is supported, only input BIRs (*ProcessedBIR*) with a *Purpose* value of BioAPI_PURPOSE_IDENTIFY must be accepted. If another purpose value is specified and not supported, an error condition shall be set as a BioAPI_RETURN.

Only the nearest, better supported *MaxFMRRequested* must be supported. Return of matching *Candidates* is required; however, the BSP may return values for the *FMRAchieved* field as next nearest step/increment. (The BSP shall indicate in component registry whether it implements coarse scoring.)

NOTE See clause C.4 for and explanation of use of FMR for scoring and thresholding.

Support of binning is optional.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.1.9 ***BioSPI_Decide*** (BioAPI2.2). If this function is supported, only score BIR produced by a call to ***BioAPI_VerifyMatchUsingAuxBIRs*** or ***BioAPI_Fuse*** shall be accepted. If another BIR is set, an error condition shall be set as a BioAPI_RETURN.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList,* corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.1.10 ***BioSPI_Fuse*** (BioAPI2.2). If this function is supported, only input BIRs of the same type (processed, score, or decision BIRs) shall be accepted. If BIRs of different types are set, an error condition shall be set as a BioAPI_RETURN.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList,* corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.2 Database operations. BSPs are not required to provide an internal (or BSP-controlled) BIR database. If one is provided, however, ALL database functions shall be provided in order to access and maintain it. All provided database functions shall conform to the definitions.

A.4.6.1.3 BioSPI_Import. This function, as a whole, is optional. If it is provided, the component registry shall so reflect, and it shall be implemented as defined.

Verification BSPs are only required to process imported data if the purpose value is BioAPI_PURPOSE_ENROLL' or BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY. Identification BSPs are only required to process imported data if the purpose value with any Enroll purpose value.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input *OutputFormat* and return the *ConstructedBIR* in that format.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.4 ***BioSPI_Export*** (BioAPI2.2). This function, as a whole, is optional. If it is provided, the componen registry shall so reflect, and it shall be implemented as defined.

Verification BSPs are only required to process the input BIR if the purpose value is BioAPI_PURPOSE_ENROLL or BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY.Identification BSPs are only required to process the input BIR if the purpose value with any Enroll purpose value.

If a BSP supports more than one output BIR data format (as indicated in the BSP schema in the component registry) it shall accept the input OutputBDBFormat and return the BIR in that format.

If ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater) has been called, security operations shall be done according to the value set in the member of the parameter *SecurityProfileList*, corresponding to the BioAPI Unit which executes this function, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

A.4.6.1.5 BioSPI_PresetIdentifyPopulation. This function, as a whole, is optional. If it is provided, the component registry shall so reflect, and it shall be implemented as defined.

A.4.6.1.6 SPI BioAPI Unit operations. All BioAPI Unit operations, as a whole, are optional. If any are provided, the component registry shall so reflect, and each function shall be implemented as defined.

A.4.6.2 Optional Subfunctions

The following table identifies optional capabilities, each of which the BSP shall declare as being supported or not supported (in the options mask within the BSP Schema in the component registry and in the BSP documentation).

Table A.2 — Optional Subfunctions

Capability	Supported	Not Supported
Return of raw/audit data		
Return of quality		
Application-controlled GUI		
GUI streaming callbacks (BioAPI 2.0)		
Generation of GUI progress events during capture suboperations (BioAPI 2.1)		
Generation of GUI progress events during identify match suboperations (BioAPI 2.1)		
Detection of source presence		
Payload		
Digital signature of BIR		
MAC generation of BIR		
Encryption of BDB		
ACBio generation		
Template update		
Adaptation		
Binning		
Self-contained device		
Match-on-Card		
Subtype to capture		
Sensor BFPs		
Archive BFPs		
Matching BFPs		
Processing BFPs		
Coarse scoring		

A.4.6.2.1 ***Return of raw data.*** Functions involving the capture of biometric data from a sensor may optionally support the return of this raw data for purposes of display or audit. If supported, the output parameter *AuditData* will contain a pointer to this data. If not supported, the BSP will return a value of BioAPI_UNSUPPORTED_BIR_HANDLE.

A.4.6.2.2 ***Return of quality.*** Upon the new capture of biometric data from a sensor, the BSP may calculate a relative quality value associated with this data, which it will include in the header of the returned *CapturedBIR* (and the optional *AuditData*). If supported, this header field will be filled with a positive value between '1' and '100'. If not supported, this field will be set to '–2'. This would occur during ***BioSPI_Capture*** and ***BioSPI_Enroll***.

Similarly, when a BIR is processed, another quality calculation may be performed and the quality value included in the header of the *ProcessedBIR* (and the optional *AdaptedBIR*). This would occur during ***BioSPI_CreateTemplate***, ***BioSPI_Process***, ***BioSPI_ProcessWithAuxBIR***, ***BioSPI_Verify***, ***BioSPI_VerifyMatch***, ***BioSPI_Enroll***, and ***BioSPI_Import*** (*ConstructedBIR*) operations.

The BSP shall post to the component registry (OptionsMask) whether or not it supports the calculation of quality measurements for each type of BIR – raw, intermediate, and processed.

NOTE See clause 7.73 for additional information regarding use of quality data.

A.4.6.2.3 ***Application-controlled GUI***. The BSP supports the necessary callbacks to allow the application to control the "look-and-feel" of the GUI.

NOTE 1 This is not applicable for BSPs designed for framework-free systems.

All BioAPI conformant BSPs shall provide, as a default, the capability to display user interface information (assuming such a display is present and required by the biometric technology) during capture operations. All BSP supplied GUIs shall include an operator abort/cancel mechanism.

Optionally, the BSP may also support application-controlled GUI. In this case, the BSP shall support the ***BioSPI_SetGUICallbacks*** function.

If application-controlled GUI is supported, the BSP may additionally provide streaming data (e.g., for voice and face recognition). If streaming data is provided, the BSP shall support the input parameters *GuiStreamingCallback* and *GuiStreamingCallbackCtx*.

All BSPs shall post to the component registry what GUI functions/options it supports.

NOTE 2 See clause C.8 for additional information regarding user interface considerations.

A.4.6.2.4 ***GUI streaming callbacks.*** The BSP provides GUI streaming data, and supports the GUI streaming callback.

NOTE This is not applicable for BSPs designed for framework-free systems.

A.4.6.2.5 ***Detection of source presence.*** The BSP can detect when there are samples available, and supports the source present event.

NOTE 1 This is not applicable for BSPs designed for framework-free systems.

NOTE 2 An example of 'samples available' would be when a finger has made physical contact with the platen of a fingerprint scanning sensor.

A.4.6.2.6 ***Payload carry.*** Some BSPs may optionally support the storage of a 'payload' when a reference template is created, and the subsequent release of that payload upon successful verification. The content of the payload is unrestricted, but may include secrets such as PINs or private keys associated with the user whose biometric data is contained within the BIR. Payload data is provided during a ***BioSPI_Enroll*** or ***BioSPI_CreateTemplate*** operation and is released as a result of a ***BioSPI_Verify*** or ***BioSPI_VerifyMatch*** operation. If supported, BSPs shall post to the component registry the maximum payload size it can accommodate. A maximum size of '0' indicates payload carry is not supported. If input payloads exceed this size, an error shall be generated. If payload carry is not supported, the output *Payload* is set to NULL.

NOTE See clause C.5 for additional information on payloads.

A.4.6.2.7 ***Security features.*** The BioAPI supports but does not require the implementation of security features. Security operations shall be done according to the value set in the member of the parameter *SecurityProfileList,* corresponding to the BioAPI Unit, of ***BioSPI_BSPAttachSecure*** (BioAPI 2.2 or greater).

The following features are optionally supported:

— Encryption of BDB;

— Digital signature of BIR (the concatenation of SBH and BDB to be exact);

— MAC generation of BIR (the concatenation of SBH and BDB to be exact);

— ACBio generation.

NOTE Key management associated with this capability is addressed by this API but the detail of key management is not described in this document. For details of key management, see RFC 3852.

A.4.6.2.8 ***Template update.*** The ***CreateTemplate*** and ***Enroll*** functions may be provided by the application with a previously enrolled template (*ReferenceTemplate*) for update (i.e., the BSP may use this template along with the newly captured sample to create the *NewTemplate*). The BSP shall in all cases return a *NewTemplate* (except in the event of an error), but if template update is not supported, it may ignore the provided *ReferenceTemplate* in its calculations.

A.4.6.2.9 ***Template adaptation***. Some BSPs may optionally provide the capability to utilize newly captured biometric data to update a reference BIR. This may only be performed as a result of a successful ***BioSPI_Verify*** or ***BioSPI_VerifyMatch*** (i.e., one in which *Result* = BioAPI_TRUE). This is performed in order to keep the enrolled BIR as fresh as possible, with the highest possible quality. The BSP makes the decision as to when and if the adaptation should be performed (based on such factors as quality, elapsed time, and significant differences). If the BSP does not support adaptation, the returned *AdaptedBIR* will be set to '–2' (BioAPI_UNSUPPORTED_BIR_HANDLE). If the BSP supports adaptation, but for some reason is not able or chooses not to perform the adaptation, then the return *AdaptedBIR* is set to '–1' (BioAPI_INVALID_BIR_HANDLE). The BSP shall post to the component registry (OptionsMask) its support for adaptation.

A.4.6.2.10 ***Binning*.** Identification BSPs may optionally support methods of limiting the population of a database to be searched in order to improve response time. This applies to Identification type BSPs only and occurs only during ***BioSPI_Identify*** and ***BioSPI_IdentifyMatch*** operations. Identification BSPs shall post whether or not they support binning. Identification BSPs that do not support binning may ignore the input *Binning* on/off parameter.

NOTE No explicit API support is provided for setting or modifying the binning strategy other than for turning it on or off.

A.4.6.2.11 ***Self-contained Device*.** The supported sensor device is "self-contained". That is, in addition to the sensor itself, at least the verification function is entirely contained within the device, and the device may also contain a BIR database.

NOTE Self-contained devices can comprise two or more BioAPI Units (i.e., the sensor unit plus one other category of BioAPI Unit).

A.4.6.2.12 ***Match-on-Card***. The BSP supports the ability for the matching operation to be executed on a smartcard. Depending on the sub-class of BSP, this requires that the BSP be able to create and operate upon BIRs suitable for MOC. It may further require that the BSP be able to interact (directly or indirectly) with the card.

A.4.6.2.13 ***Subtype to capture***. The BSP supports the ability for the application to specify which biometric subtype(s), of a given biometric type, to capture (e.g., the right index finger or left eye). If this option is supported and if the application specifies a subtype to capture, then the BSP will capture that specific biometric feature and so indicate in the header of the returned *CapturedBIR*. This applies to all functions requiring a biometric capture (i.e., ***BioSPI_Capture***, ***BioSPI_Enroll***, ***BioSPI_Verify***, or ***BioSPI_Identify***).

A.4.6.2.14 ***Sensor BFP***. The BSP supports the attachment of alternative sensor devices (units) via an associated FPI. When attached, the BSP will utilize that sensor unit for any functions requiring a biometric capture (i.e., ***BioSPI_Capture***, ***BioSPI_Enroll***, ***BioSPI_Verify***, or ***BioSPI_Identify***).

A.4.6.2.15 ***Archive BFP***. The BSP, in addition to any internal archive capability, supports the attachment of an alternative archive unit. The BSP thus supports the interface to such a unit through the associated FPI. This would also require that the BSP support the BioSPI database operations. When attached, the

BSP utilizes this archive when the Database UUID associated with this archive (BIR database) is specified for any database function.

A.4.6.2.16 ***Matching BFP.*** The BSP supports the attachment of an alternative matching unit (algorithm). When attached, the BSP will utilize the alternate unit for any matching operation for which it is capable (i.e., for verify operations if it is a 1:1 matching module).

A.4.6.2.17 ***Processing BFP.*** The BSP supports the attachment of an alternative processing unit (algorithm). When attached, the BSP will utilize the alternate unit for any ***BioSPI_Process*** or ***BioSPI_CreateTemplate*** calls.

A.4.6.2.18 ***Coarse scoring***. The BSP supports the return of scores in increments that have purposely been increased to counter specific attack scenarios (see clause C.4.6). Unless this option is indicated in the OptionsMask, the BSP provides finely quantized scores.

Annex B
(normative)

CBEFF Patron Format Specification: BioAPI patron format

B.1 Patron

ISO/IEC JTC1 SC 37

B.2 Patron identifier

`257 (0x0101)`. This has been allocated by the Registration Authority.

B.3 Patron format name

`ISO/IEC JTC1 SC37 BioAPI patron format.`

B.4 Patron format identifier

`6 (0x0006)`. This has been registered by the Registration Authority.

B.5 ASN.1 object identifier for this patron format

{ISO registration-authority cbeff(19785) biometric-organization(0) jtc1-sc37(257) patron-format(1) bioAPI (8)}}

or, in XML value notation,

`1.1.19785.0.257.1.6`

B.6 Domain of Use

Applications and BSPs requiring a BIR that is fully supported by the C structures specified in this document.

B.7 Version identifier

This patron format specification has a version identifier of `{major(2) minor(0)}, or 0x20.`

B.8 CBEFF version

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

B.9 General

This patron format is consistent with the normative specification of the BioAPI C data-structures (and values of these structures) for CBEFF data elements, as specified in Clause 7.

This patron format is intended to be used when the BioAPI BIR is persistently stored or transmitted as a unit between systems.

NOTE Annex D provides sample code for creating a BioAPI Patron Format from a BioAPI_BIR C structure as defined in clause 7.7.

The byte/bit order of implementations of this format shall be big-endian, with no padding between data elements, for all storage and transfer between systems.

B.10 Specification

CBEFF-defined data elements that are not included in this table are specified as having the abstract value NO VALUE AVAILABLE in this patron format.

BioAPI field name, BioAPI reference, and patron format field name	CBEFF data element name	Length (bytes)	Abstract values	Encoding[a]	CBEFF Reference
BIR Length[b]	n/a	4[c]	Length in bytes of the entire BIR encoding, with the exception of this length field.	Integer	n/a
BioAPI_VERSION (BioAPI 7.85)	CBEFF_patron_header_version	1	Major = 2 Minor = 0	'20'	6.5.24
BioAPI_BIR_DATA_TYPE (BioAPI 7.12)	CBEFF_BDB_encryption_options	1	NOT ENCRYPTED ENCRYPTED	'0_' '1_'	6.5.3
	CBEFF_BIR_integrity_options		NOT SIGNED SIGNED	'0_' '2_'	6.5.4
	CBEFF_BDB_processed_level		NO VALUE AVAILABLE RAW INTERMEDIATE PROCESSED	'_0' '_1' '_2' '_4'	6.5.11
(index flag)	n/a		INDEX PRESENT	'8_'	n/a
BioAPI_BIR_BIOMETRIC_DATA_FORMAT (BioAPI 7.9)	CBEFF_BDB_format_owner CBEFF_BDB_format_type	2 2	FormatOwner FormatType	Integer integer	6.5.1 6.5.2
BioAPI_QUALITY (BioAPI 7.73)	CBEFF_BDB_quality	1	NOT SUPPORTED NOT SET (NO VALUE AVAILABLE) VALUE	-2 -1 0-100	6.5.15
BioAPI_BIR_PURPOSE (BioAPI 7.16)	CBEFF_BDB_purpose	1	NO VALUE AVAILABLE VERIFY IDENTIFY ENROLL ENROLL_FOR_VERIFICATION_ONLY ENROLL_FOR_IDENTIFICATION_ONLY AUDIT	0 1 2 3 4 5 6	6.5.14