INTERNATIONAL STANDARD

ISO/IEC 30106-2

First edition
2016-03-15

Information technology — Object oriented BioAPI —

Part 2: Java implementation

Technologies de l'information — Objet orienté BioAPI —

Partie 2: Mise en oeuvre Java

Reference number
ISO/IEC 30106-2:2016(E)

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

Contents

Page

Foreword

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

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 meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/TC JTC1, *Information technology*, Subcommittee SC 37, *Biometrics*.

ISO/IEC 30106 consists of the following parts, under the general title *Information technology — Object oriented BioAPI*:

— *Part 1: Architecture*

— *Part 2: Java implementation*

— *Part 3: C# implementation*

Introduction

In this part of ISO/IEC 30106, an application programming interface expressed in Java language is specified. Java is intended to be a simple, general-purpose, object oriented programming language that is aimed at enabling programmers to quickly build a wide range of applications for multiple platforms.

This Java implementation allows an easy use of Java BSPs, Java-based application servers or Java applets. Therefore, it is the best way to write desktop and web applications/services and this specification provides an advanced and well-designed remote framework.

Although the best practices of Java programming states that variables should be written in smallcase letters, in the case of symbols, such as BSP or BFPs, it has been kept as uppercase letters.

Information technology — Object oriented BioAPI —

Part 2:
Java implementation

1 Scope

This part of ISO/IEC 30106 specifies an interface of a BioAPI Java framework and BioAPI Java BSP, which will mirror the corresponding components, specified in ISO/IEC 30106-1. The semantic equivalent of this standard is maintained in this part of ISO/IEC 30106.

2 Normative references

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

ISO/IEC 30106–1, *Information technology — BioAPI for object oriented programming languages — Part 1: Architecture*

3 BioAPI Java package structure

The BioAPI Java interface will be divided into several packages. The following is the package structure:

— package org.bioapi: contains functionality to manage units, BSPs, BFPs, the Framework and Applications;

— package org.bioapi.data: contains all the data structures.

3.1 Package org.bioapi

3.1.1 Package description

This package contains all the components responsible for managing and executing the functionality of BioAPI. Component Registry interface is also defined in this package.

3.1.2 Structure

The description of this namespace is given explaining a bottom-up structure. In Clause 4, the interfaces needed to be implemented for each of the Unit types are explained. It is important to note that such interfaces do not refer to an implemented class by itself, as the accessible class will either be the Biometric Service Provider (BSP) or the Biometric Function Provider (BFP), but the specifications in such clause are common to the methods and properties to be added to the implemented BSP and/or BFP classes.

This will be followed by the specification of the implementation of the BFP (Clause 5) and BSP (Clause 6) interfaces. These two interfaces provide the lower layer interoperability level, equivalent to the SPI and BFPI interfaces in ISO/IEC 19784-1.

The higher layer of interoperability level is provided by the specification of the Framework (Clause 7, with the Framework Interface and the Component Registry) and the Application interaction (Clause 8,

with the specification of the Exceptions and Callback functions). This provides the equivalence to the API interface in ISO/IEC 19784-1.

3.2 Package org.bioapi.data

3.2.1 Package description

This package contains all data structures needed for the implementation of OO BioAPI.

3.2.2 Structure

Several data structures are provided to comply with the requirements specified in this part of ISO/IEC 30106. All the BioAPI.Data namespace is specified in Clause 3, where all needed classes and enumerations are defined. This has to be complemented to the constants defined in ISO/IEC 30106-1.

4 Data types and constants

4.1 Class ACBioParameters

4.1.1 Description

Structure that provides the information that is used to generate ACBio instances.

4.1.2 Method summary

4.1.2.1	**int[] getChallenge()**
Description:	Return the challenge from the validator of a biometric verification when ACBio is used. This value shall be sent to the field controlValue of type ACBioContentInformation in ACBio instances.
Return value:	The challenge from the validator of a biometric verification when ACBio is used.

4.1.2.2	**int[] getInitialBPUIOIndexOutput()**
Description:	Return 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 inside the BSP and assigned to the BSP Units and BSPs.
Return value:	The initial value of BPU IO index.

4.1.2.3	**int[] getSupremumBPUIOIndexOutput()**
Description:	Return 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.
Return value:	The supremum of BPU IO index.

4.2 Class BFPListElement

4.2.1 Description

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

4.2.2 Method summary

4.2.2.1	**UUID getBFPID()**
Description:	Return the UUID assigned to the BFP.
Return value:	UUID assigned to the BFP.

4.2.2.2	**UnitCategoryType getUnitCategory()**
Description:	Return the category of the units.
Return value:	Category of the units.

4.2.2.3	**void setBFPID(UUID bfpID)**
Description:	Set the UUID assigned to the BFP.
Parameters:	*bfpID:* UUID assigned to the BFP.

4.2.2.4	**void setUnitCategory(UnitCategoryType unitCategory)**
Description:	Set the category of the units.
Parameters:	*unitCategory:* category of the units.

4.3 Class BFPSchema

4.3.1 Description

Represents the record in the component registry that defines the properties of the BFP installed in the system.

4.3.2 Method summary

4.3.2.1	**String getBFPDescription()**
Description:	Return a string containing a text description of the BFP.
Return value:	A string containing a text description of the BFP.

4.3.2.2	**Vector<RegistryID> getBFPSupportedFormats()**
Description:	Return a list the data formats that are supported by the BFP.
Return value:	A list the data formats that are supported by the BFP.

4.3.2.3	**UUID getBFPUUID()**
Description:	Return the BFP UUID
Return value:	BFP UUID

4.3.2.4	**Vector<BiometricType> getFactorsMask()**
Description:	Return a list of the biometric types supported by the BFP.
Return value:	A list of the biometric types supported by the BFP.

4.3.2.5	**byte[] getFWProperty(**
Description:	Return the 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.
Return value:	The address and length of a memory buffer containing the BFP property.

4.3.2.6	UUID getFWPropertyID()
Description:	Return the UUID of the format of the following BFP property.
Return value:	UUID of the format of the following BFP property.

4.3.2.7	String getPath()
Description:	Return a pointer to a string containing the path of the file containing the BFP 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 10646:2014, Annex D). When BFPSchema 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.
Return value:	A pointer to a string containing the path of the file containing the BFP executable code, including the filename.

4.3.2.8	String getProductVersion()
Description:	Return the version string of the BFP software.
Return value:	The version string of the BFP software.

4.3.2.9	String getSpecVersion()
Description:	Return the major/minor version number of the BioAPI specification to which the BFP was implemented.
Return value:	The major/minor version number of the BioAPI specification to which the BFP was implemented.

4.3.2.10	UnitCategoryType getUnitCategory()
Description:	Return the category of the BFP identified by the BFP UUID.
Return value:	The category of the BFP identified by the BFPUUID.

4.3.2.11	String getVendor()
Description:	Return a string containing the name of the BFP vendor.
Return value:	A string containing the name of the BFP vendor.

4.4 Class BIR

4.4.1 Description

This interface represents Biometric Information Records (BIRs). It supports ISO/IEC 19785 definitions, both for Simple-BIRs or for Complex-BIRs. The specification of the patron format that shall be used is given in ISO/IEC 30106-1.

4.4.2 Method summary

4.4.2.1	void birFromByteArray(byte[] record)
Description:	Fills in the BIR data from a byte array coded as ISO/IEC 19785 record.
Parameters:	*record*: the byte array containing the CBEFF record.
Exception:	If the input parameters are invalid, the format is not supported or operation fails due to error. BioAPIException (see 9.1)

4.4.2.2	byte[] birToByteArray()
Description:	Serializes a BIR record so as to provide it as a byte array representing the CBEFF information.
Return value:	The byte array containing the CBEFF information.
Exception:	If the input parameters are invalid, the format is not supported or operation fails due to error. BioAPIException (see 9.1)

4.4.2.3	void destroy()
Description:	Removes all the information in the current BIR, leaving it empty for a next use.
Exception:	None

4.4.2.4	BiometricSubtype getBDBBiometricSubtype()
Description:	Return the BDB biometric subtype
Return value:	The BDB biometric subtype

4.4.2.5	BiometricType getBDBBiometricType()
Description:	Return the BDB biometric type
Return value:	The BDB biometric type

4.4.2.6	byte[] getBDBChallengeResponse()
Description:	Return the BDB challenge response
Return value:	The BDB challenge response

4.4.2.7	Date getBDBCreationDate()
Description:	Return the BDB creation date
Return value:	The BDB creation date

4.4.2.8	byte[] getBDBData()
Description:	Return the BDB data array
Return value:	The BDB data array

4.4.2.9	RegistryID getBDBFormat()
Description:	Return the format of the BDB data
Return value:	The format of the BDB data

4.4.2.10	byte[] getBDBIndex()
Description:	Return the BDB index
Return value:	The BDB index

4.4.2.11	ProcessedLevel getBDBProcessedLevel()
Description:	Return the BDB processed level
Return value:	The BDB processed level

4.4.2.12	**Purpose getBDBPurpose()**
Description:	Return the BDB purpose
Return value:	The BDB purpose

4.4.2.13	**byte getBDBQuality()**
Description:	Return the BDB quality
Return value:	The BDB quality

4.4.2.14	**Vector<Date> getBDBValidityPeriod()**
Description:	Return the BDB validity period
Return value:	The BDB validity period

4.4.2.15	**Date getBIRCreationDate()**
Description:	Return the BIR creation date
Return value:	The BIR creation date

4.4.2.16	**byte[] getBIRCreator()**
Description:	Return the BIR creator array
Return value:	The BIR creator array

4.4.2.17	**byte[] getBIRIndex()**
Description:	Return the BIR index
Return value:	The BIR index

4.4.2.18	**byte[] getBIRAdditionalData()**
Description:	Return the BIR additionalData
Return value:	The BIR additionalData

4.4.2.19	**Vector<Date> getBIRValidityPeriod()**
Description:	Return the BIR validity period
Return value:	The BIR validity period

4.4.2.20	**byte getCBEFFVersion()**
Description:	Return the version of the CBEFF component
Return value:	The version of the CBEFF component

4.4.2.21	**RegistryID getPatronFormat()**
Description:	Return the patron format
Return value:	The patron format

4.4.2.22	**byte getPatronHeaderVersion()**
Description:	Return the header version
Return value:	The header version

4.4.2.23	**byte[] getSBData()**
Description:	Return the Security Block data array
Return value:	The Security Block data array

4.4.2.24	**RegistryID getSBFormat()**
Description:	Return the Security Block format
Return value:	The Security Block format

4.4.2.25	**boolean hasBDBEncription()**
Description:	Return true if the BDB is encrypted, false otherwise.
Return value:	True if the BDB is encrypted, false otherwise.

4.4.2.26	**boolean hasBDBIntegrity()**
Description:	Return true if the BDB has integrity, false otherwise.
Return value:	True if the BDB has integrity, false otherwise.

4.4.2.27 4.4.2.28 4.4.2.29	**boolean isBIRSigned()** **boolean isQualitySupported()** **boolean isQualityKnown()**
Description:	Request about each of the above mentioned characteristics of the BIR.
Return value:	True if the characteristic is available, false in other case.
Exception:	None

4.4.2.30	**void setBDBBiometricSubtype(BiometricSubtype bdbBiometricSubtype)**
Description:	Set the BDB biometric subtype
Parameters:	*bdbBiometricSubtype*: BDB biometric subtype

4.4.2.31	**void setBDBBiometricType(BiometricType bdbBiometricType)**
Description:	Set the BDB biometric type
Parameters:	*bdbBiometricType*: BDB biometric type

4.4.2.32	**void setBDBChallengeResponse(byte bdbChallengeResponse)**
Description:	Set the BDB challenge response
Parameters:	*bdbChallengeResponse*: BDB challenge response

4.4.2.33	**void setBDBCreationDate(Date bdbCreationDate)**
Description:	Set the BDB creation date
Parameters:	*bdbCreationDate*: BDB creation date

4.4.2.34	**void setBDBEncription(boolean bdbEncription)**
Description:	Determine if BDB data is encrypted or not
Parameters:	*bdbEncription*: true if the BDB is encrypted, false otherwise.

4.4.2.35	**void setBDBFormat(RegistryID bdbFormat)**
Description:	Set the format of the BDB data
Parameters:	*bdbFormat*: format of the BDB data

4.4.2.36	**void setBDBData(byte[] bdbData)**
Description:	Set the BDB data
Parameters:	*bdbData*: BDB data

4.4.2.37	**void setBDBIndex(byte[] bdbIndex)**
Description:	Set the BDB index array
Parameters:	*bdbIndex*: BDB index array

4.4.2.38	**void setBDBIntegrity(boolean bdbIntegrity)**
Description:	Determine if BDB data has integrity or not
Parameters:	*bdbIntegrity*: true if the BDB has integrity, false otherwise.

4.4.2.39	**void setBDBQuality(byte bdbQuality)**
Description:	Set the BDB quality
Parameters:	*bdbQuality*: BDB quality

4.4.2.40	**void setBDBProcessedLevel(ProcessedLevel bdbProcessedLevel)**
Description:	Set the BDB processed level
Parameters:	*bdbProcessedLevel*: BDB processed level

4.4.2.41	**void setBDBPurpose(Purpose bdbPurpose)**
Description:	Set the BDB purpose
Parameters:	*bdbPurpose*: BDB purpose

4.4.2.42	**void setBDBValidityPeriod(Vector<Date> bdbValidityPeriod)**
Description:	Set the BDB validity period
Parameters:	*bdbValidityPeriod*: BDB validity period

4.4.2.43	**void setBIRCreationDate(Date birCreationDate)**
Description:	Set the BIR creation date
Parameters:	*birCreationDate*: BIR creation date

4.4.2.44	**void setBIRCreator(byte[] birCreator)**
Description:	Set the BIR creator array
Parameters:	*birCreator*: BIR creator array

4.4.2.45	**void setBIRIndex(byte[] birIndex)**
Description:	Set the BIR index
Parameters:	*birIndex*: BIR index

4.4.2.46	**void setBIRAdditionalData(byte[] birAdditionalData)**
Description:	Set the BIR additionalData
Parameters:	*birAdditionalData*: BIR additionalData

4.4.2.47	**void setBIRValidityPeriod(Vector<Date> birValidityPeriod)**
Description:	Set the BIR validity period
Parameters:	*birValidityPeriod*: BIR validity period

4.4.2.48	**void setCBEFFVersion(byte cbeffVersion)**
Description:	Set the version of the CBEFF component
Parameters:	*cbeffVersion*: version of the CBEFF component header version

4.4.2.49	**void setPatronFormat(RegistryID patronFormat)**
Description:	Set the patron format
Parameters:	*patronFormat*: patron format

4.4.2.50	**void setPatronHeaderVersion(byte patronHeaderVersion)**
Description:	Set the header version
Parameters:	*patronHeaderVersion*: header version

4.4.2.51	**void setSBData(byte[] sbData)**
Description:	Set the SB data
Parameters:	*sbData*: SB data

4.4.2.52	**void setSBFormat(RegistryID sbFormat)**
Description:	Set the format of the SB data
Parameters:	*sbFormat*: format of the SB data

4.5 Class BSPSchema

4.5.1 Description

Represents the record in the component registry that defines the properties of the BSP installed in the system.

4.5.2 Method summary

4.5.2.1	**UUID getBSPAccessUUID()**
Description:	Return 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 part of ISO/IEC 30106 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.
Return value:	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.

4.5.2.2	String getBSPDescription()
Description:	Return a string containing a text description of the BSP.
Return value:	A string containing a text description of the BSP.

4.5.2.3	Vector<RegistryID> getBSPSupportedAlgorithms()
Description:	Return an array of BioAPI_ALGORITHM_ID structures specifying the supported algorithms.
Return value:	Array of BioAPI_ALGORITHM_ID structures specifying the supported algorithms.

4.5.2.4	Vector<RegistryID> getBSPSupportedFormats()
Description:	Return a list the data formats that are supported by the BSP.
Return value:	A list the data formats that are supported by the BSP.

4.5.2.5	Vector<UUID> getBSPSupportedTransformOperation()
Description:	Return an array of BioAPI_UUID structures specifying the transform operations supported within the BioAPI_Tranform operation.
Return value:	Array of BioAPI_UUID structures specifying the transform operations supported within the BioAPI_Tranform operation.

4.5.2.6	UUID getBSPUUID()
Description:	Return the BSP UUID
Return value:	The BSP UUID

4.5.2.7	int getDefaultCalibrateTimeout()
Description:	Return the default timeout value in milliseconds used by the BSP for Calibrate operations when no timeout is specified by the application.
Return value:	The default timeout value for Calibrate operations.

4.5.2.8	int getDefaultCaptureTimeout()
Description:	Return the default timeout value in milliseconds used by the BSP for Capture operations when no timeout is specified by the application.
Return value:	The default timeout value for Capture operations.

4.5.2.9	int getDefaultEnrolTimeout()
Description:	Return the default timeout value in milliseconds used by the BSP for Enrol operations when no timeout is specified by the application.
Return value:	The default timeout value for Enrol operations.

4.5.2.10	int getDefaultIdentifyTimeout()
Description:	Return the default timeout value in milliseconds used by the BSP for Identify operations when no timeout is specified by the application.
Return value:	The default timeout value for Identify operations.

4.5.2.11	int getDefaultVerifyTimeout()
Description:	Return the default timeout value in milliseconds used by the BSP for Verify operations when no timeout is specified by the application.
Return value:	The default timeout value for Verify operations.

4.5.2.12	Vector<BiometricType> getFactorsMask()
Description:	Return a list of the biometric types supported by the BSP.
Return value:	A list of the biometric types supported by the BSP.

4.5.2.13	byte[] getHostingEndpointIRI()
Description:	Return an IRI identifying the framework whose component registry contains a registration of the BSP. This parameter shall be ignored by frameworks conforming to this part of ISO/IEC 30106 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.
Return value:	An IRI identifying the framework whose component registry contains a registration of the BSP.

4.5.2.14	int getMaxBSPDbSize()
Description:	Return the maximum size of a BSP-controlled BIR database. It applies only when a BSP is only capable of directly managing a single archive unit. 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.
Return value:	The maximum size of a BSP-controlled BIR database.

4.5.2.15	int getMaxIdentify()
Description:	Return the largest population supported by the identify function. Unlimited = 0xFFFFFFFF.
Return value:	The largest population supported by the identify function.

4.5.2.16	int getMaxNumEnrolInstances()
Description:	Return 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.
Return value:	The maximum number of distinct instances that a BSP can create reference templates for in one enroll operation.

4.5.2.17	int getMaxAdditionalDataSize()
Description:	Return the maximum additionalData size (in bytes) that the BSP can accept
Return value:	The maximum additionalData size (in bytes) that the BSP can accept

4.5.2.18	Vector<BSPSchemaOperations> getOperations()
Description:	Return a list of the biometric operations supported by the BSP.
Return value:	A list of the biometric operations supported by the BSP.

4.5.2.19	Vector<BSPSchemaOptions> getOptions()
Description:	Return a list of the biometric options supported by the BSP.
Return value:	A list of the biometric options supported by the BSP.

4.5.2.20	String getPath()
Description:	Return a pointer to a 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 10646:2014, Annex D). 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.
Return value:	A pointer to a string containing the path of the file containing the BSP executable code, including the filename.

4.5.2.21	int getAdditionalDataPolicy()
Description:	Return the threshold setting (maximum FMR value) used to determine when to release additionalData after successful verification.
Return value:	The threshold setting (maximum FMR value) used to determine when to release additionalData after successful verification.

4.5.2.22	String getProductVersion()
Description:	Return the version string of the BSP software.
Return value:	The version string of the BSP software.

4.5.2.23	String getSpecVersion()
Description:	Return the major/minor version number of the BioAPI specification to which the BSP was implemented.
Return value:	Major/minor version number of the BioAPI specification to which the BSP was implemented.

4.5.2.24	String getVendor()
Description:	Return a string containing the name of the BSP vendor.
Return value:	A string containing the name of the BSP vendor.

NOTE 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 ISO/IEC 19784-1).

4.6 Class candidate

4.6.1 Description

Defines each of the resulting candidates from the Identify functionality.

4.6.2 Method summary

4.6.2.1	int getFMRAchieved()
Description:	Return a integer value which states the FMR achieved by the candidate during the Identify process.
Return value:	FMR value

4.6.2.2	UUID getKey()
Description:	Return a key value which defines the UUID of the candidate in the system (e.g. in the database).
Return value:	Key value

4.6.2.3	void setFMRAchieved(int fmrAchieved)
Description:	Set the FMR achieved by the candidate during the Identify process.
Parameters:	*fmrAchieved*: FMR value

4.6.2.4	void setKey(UUID key)
Description:	Set the key value which defines the UUID of the candidate in the system (e.g. in the database).
Parameters:	*key*: key value

4.7 Class DataTypes

4.7.1 Description

This class defines the different data types that can be used throughout this part of ISO/IEC 30106. It embraces enumerations and constants. Values for the constants not defined in this part of ISO/IEC 30106 are defined in ISO/IEC 30106-1.

4.7.2 Enumerations

4.7.2.1	BiometricSubtype
Description:	Subtype of the biometric data used (e.g. which finger used in finger modalities). When transferring this information into/from a binary format, the Biometric Subtype constants defined in ISO/IEC 30106-1 shall be used.
Enum constant summary:	— *NO_VALUE_AVAILABLE* — *LEFT* — *RIGHT* — *LEFT_THUMB* — *LEFT_INDEX_FINGER* — *LEFT_MIDDLE_FINGER* — *LEFT_RING_FINGER* — *LEF_TLITTLE_FINGER* — *RIGHT_THUMB* — *RIGHT_INDEX_FINGER* — *RIGHT_MIDDLE_FINGER* — *RIGHT_RING_FINGER* — *RIGHT_LITTLE_FINGER* — *LEFT_PALM* — *LEFT_BACK_OF_HAND* — *LEFT_WRIST* — *RIGHT_PALM* — *RIGHT_BACK_OF_HAND* — *RIGHT_WRIST*

4.7.2.2	**BiometricType**
Description:	Type of the biometric data used (e.g. modality). When transferring this information into/from a binary format, the Biometric Type constants defined in ISO/IEC 30106-1 shall be used.
Enum constant summary:	— *NO_VALUE_AVAILABLE* — *MULTIPLE_BIOMETRIC_TYPES* — *FACE* — *VOICE* — *FINGER* — *IRIS* — *RETINA* — *HAND_GEOMETRY* — *SIGNATURE_OR_SIGN* — *KEYSTROKE* — *LIP_MOVEMENT* — *GAIT* — *VEIN* — *DNA* — *EAR* — *FOOT* — *SCENT*

4.7.2.3	**BIRDatabaseAccess**
Description:	Defines the access mode to the database.
Enum constant summary:	— *READ*: access mode which allows only retrieval of records. — *READ_WRITE*: access mode which allows addition, deletion and retrieval of records. — *WRITE*: access mode which allows addition and deletion of records, but retrieval is not allowed.

4.7.2.4	**BSPSchemaOperations**
Description:	Enumerates the different operations that a BSP can offer to the biometric application (see 4.5).
Enum constant summary:	— *CALIBRATE_SENSOR (0x00020000)* — *CAPTURE (0x00000004)* — *CHECK_QUALITY (0x00080000)* — *CONTROL_UNIT (0x00400000)* — *CREATE_TEMPLATE (0x00000008)* — *CREATE_TEMPLATE_WITH_AUX_BIR (0x00000020)* — *ENABLE_EVENTS (0x00000001)* — *ENROL (0x00000100)* — *GET_INDICATOR_STATUS (0x00010000)* — *IDENTIFY (0x00000080)* — *IDENTIFY_AGGREGATE (0x00000400)* — *PRESET_IDENTIFY_POPULATION (0x00001000)* — *PROCESS (0x00000010)* — *PROCESS_WITH_AUX_BIR (0x01000000)* — *QUERY_BFPS (0x00200000)* — *QUERY_UNITS (0x00100000)* — *SECURITY (0x10000000)* — *SET_INDICATOR_STATUS (0x00008000)* — *SET_POWER_MODE (0x00004000)* — *VERIFY (0x00000040)* — *VERIFY_AGGREGATED (0x00000200)* — *VERIFY_WITH_AUX_BIR (0x02000000)*

4.7.2.5	**BSPSchemaOptions**
Description:	Enumerates the different options that can a handle a BSP (see 4.5).
Enum constant summary:	— *ADAPTATION (0x00000800)* — *APP_GUI (0x00000010)* — *ACHIVE_BFP (0x00020000)* — *BINNING (0x00001000)* — *BIR_ENCRYPT (0x00000200)* — *BIR_SIGN (0x00000100)* — *CAPTURE_MULTIPLE (0x00400000)* — *COARSE_SCORES (0x00100000)* — *COMPARISON_BFP (0x00040000)* — *DISABILITIES* — *GUI_PROGRESS_EVENTS (0x00000020)* — *IDENTIFY_INDICATOR (0x00200000)* — *OCC (0x00004000) (formerly known as MOC)* — *PAD_FEATURE* — *PAYLOAD (0x00000080)* — *PROCESSING_BFP (0x00080000)* — *PROCESS_MULTIPLE (0x00800000)* — *QUALITY_INTERMEDIATE (0x00000004)* — *QUALITY_PROCESSED (0x00000008)* — *QUALITY_RAW (0x00000002)* — *RAW (0x00000001)* — *SELF_CONTAINED_DEVICE (0x00002000)* — *SENSOR_BFP (0x00010000)* — *SOURCE_PRESENT (0x00000040)* — *SUBTYPE_TO_CAPTURE (0x00008000)* — *TEMPLATE_UPDATE (0x00000400)*

4.7.2.6	**EventKind**
Description:	Defines the kind of sources that can originate an event.
Enum constant summary:	— *INSERT (0x00000001)* — *REMOVE (0x00000002)* — *FAULT (0x00000004)* — *SOURCE_PRESENT (0x00000008)* — *SOURCE_REMOVED (0x00000010)*

4.7.2.7	Facility
Description:	Defines originator of the error in a exception (see 9.1).
Enum constant summary:	— *FRAMEWORK: error was reported by the framework component.* — *BSP: error was reported by the biometric service provider.* — *UNIT: error was reported by the biometric unit*

4.7.2.8	GUIEnrolType
Description:	Indicates the enrol type of a BSP (see 9.2).
Enum constant summary:	— *TEST_VERIFY* — *MULTIPLE_CAPTURE*

4.7.2.9	GUIMoment
Description:	Determines the moment when the processing of an operation is at the time of calling a GUI callback function (see 9.2).
Enum constant summary:	— *BEFORE_START* — *AFTER_END*

4.7.2.10	GUIOperation
Description:	Determines the operation being performed when using GUI callback functions (see 9.2).
Enum constant summary:	— *CAPTURE* — *ENROL* — *IDENTIFY* — *VERIFY*

4.7.2.11	GUIResponse
Description:	Enumeration of the possible actions to be performed bya BSP after a GUI event notification callback made by the BSP has returned control to the BSP (see 9.2).
Enum constant summary:	— *CYCLE_START* — *CYCLE_RESTART* — *DEFAULT* — *OP_COMPLETE* — *OP_CANCEL* — *PROGRESS_CONTINUE* — *PROGRESS_ABORT* — *RECAPTURE* — *SUB_OP_START* — *SUB_OP_NEXT*

4.7.2.12	GUISuboperation
Description:	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 (see 9.2).
Enum constant summary:	— *CAPTURE* — *CREATE_TEMPLATE* — *IDENTIFY* — *PROCESS* — *VERIFY*

4.7.2.13	ProcessedLevel
Description:	Determines the level of processing of the BIR.
Enum constant summary:	— *INTERMEDIATE* — *PROCESSED* — *RAW*

4.7.2.14	Purpose
Description:	Defines the purpose for which the BIR or process is intended.
Enum constant summary:	— *VERIFY* — *IDENTIFY* — *ENROL* — *ENROL_FOR_VERIFICATION_ONLY* — *ENROL_FOR_IDENTIFICACION_ONLY* — *AUDIT* — *DECIDE* — *NO_PURPOSE_AVAILABLE*

4.7.2.15	ResultOptions
Description:	Defines the request to some BioAPI methods, to provide additional results to the originally defined (e.g. see 5.3.2).
Enum constant summary:	— *REQUEST_ADAPTED_BIR: request that a BIR be constructed by adapting the reference template using the processed BIR that is the input to the biometric verification.* — *REQUEST_PAYLOAD: request that the additionalData should be returned upon successful verification if the achieved.* — *REQUEST_ADDITIONAL_DATA: request additional data to be used, for example, in an auditing process.*

4.7.2.16	SecurityOptionsType
Description:	Defines which security options are supported by the BioAPI_Unit.
Enum constant summary:	— *ENCRYPTION (0x00000001): indicates that the BioAPI Unit supports encryption.* — *MAC (0x00000002): indicates that the BioAPI Unit supports MAC generation.* — *DIGITAL_SIGNATURE (0x00000004): indicates that the BioAPI Unit supports digital signature.* — *AC_BIO_GENERATION_WITH_MAC (0x00000010): indicates that the BioAPI Unit supports ACBio generation using MAC.* — *AC_BIO_GENERATION_WITH_DIGITAL_SIGNATURE (0x00000020): indicates that the BioAPI Unit supports ACBio generation using digital signature.*

4.7.2.17	UnitCategoryType
Description:	List the different categories for a BioAPI_Unit
Enum constant summary:	— *ARCHIVE*: the unit manages BSP's BIR database. (0x00000001). — *COMPARISON*: the unit is the collection of comparison algorithms. (0x00000002) — *PROCESSING*: the unit is the collection of processing algorithms. (0x00000004) — *SENSOR*: the unit manages hardware sensor (0x00000008).

4.7.2.18	UnitIndicatorStatus
Description:	Defines possible values for the indicator status
Enum constant summary:	— *ACCEPT* — *BUSY* — *FAILURE* — *READY* — *REJECT*

4.7.2.19	UnitPowerMode
Description:	Defines possible unit power modes
Enum constant summary:	— *DETECT*: *mode when unit is able to detect interaction of subject with the sensor.* — *NORMAL*: *mode when all functions are available.* — *SLEEP*: *minimum mode. All functions off.*

4.8 Class date

4.8.1 Description

Represents calendar date and time. Both date and time are coded in the same inferface as CBEFF and ISO/IEC 19794 code it in that way. Each of the properties has been defined as integers for homogenity among properties. Also, comparison methods are available to help on the use of complex date and time values.

4.8.2 Method summary

4.8.2.1	**int getDayOfMonth()**
Description:	Return the day of the month
Return value:	The day of the month

4.8.2.2	**int getHour()**
Description:	Return the hour
Return value:	The hour

4.8.2.3	**int getMinute()**
Description:	Return the minute
Return value:	The minute

4.8.2.4	**int getMonth()**
Description:	Return the month
Return value:	The month

4.8.2.5	**int getSecond()**
Description:	Return the second
Return value:	The second

4.8.2.6	**int getYear()**
Description:	Return the year
Return value:	The year

4.8.2.7 4.8.2.8 4.8.2.9 4.8.2.10 4.8.2.11 4.8.2.12	**boolean isLowerOrEqual (int day, int month, int year)** **boolean isLowerOrEqual (int day, int month, int year, int hour, int minute, int second)** **boolean isLowerOrEqual (Date date)** **boolean isHigherOrEqual (int day, int month, int year)** **boolean isHigherOrEqual (int day, int month, int year, int hour, int minute, int second)** **boolean isHigherOrEqual (Date date)**
Description:	Compares the date and time of the object within the interface, with the one represented by the parameters of called method. The name of the method refers to the following operations: — isLowerOrEqual: “date and time given by parameters” <= “object’s date and time” — isHigherOrEqual: “date and time given by parameters” >= “object’s date and time”

Parameters:	— *day*: day of the month for the date to be compared with the object's date — *month*: month for the date to be compared with the object's date — *year*: year for the date to be compared with the object's date — *hour*: hour for the date to be compared with the object's date — *minute*: minute for the date to be compared with the object's date — *second*: secondfor the date to be compared with the object's date — *date*: date to be compared with the object's date
Return value:	True if the condition is met, False in any other case.
Exception:	If the input parameters are invalid, the format is not supported or operation fails due to error. BioAPIException (see 9.1)

4.8.2.13	**void setDayOfMonth(int dayOfMonth)**
Description:	Set the day of the month
Parameters:	*dayOfMonth:* day of the month

4.8.2.14	**void setHour(int hour)**
Description:	Set the hour
Parameters:	*hour:* the hour

4.8.2.15	**void setMinute(int minute)**
Description:	Set the minute
Parameters:	*minute:* the minute

4.8.2.16	**void setMonth(int month)**
Description:	Set the month
Parameters:	*month:* the month

4.8.2.17	**void setSecond(int second)**
Description:	Set the second
Parameters:	*second:* the second

4.8.2.18	**void setYear(int year)**
Description:	Set the year
Parameters:	*year:* the year

4.9 Class FrameworkSchema

4.9.1 Description

Defines the properties of the BioAPI Framework.

4.9.2 Method summary

4.9.2.1	**UUID getFrameworkUUID()**
Description:	Return the UUID of the Framework component
Return value:	The UUID of the Framework

4.9.2.2	**String getFWDescription()**
Description:	Return a string containing a text description of the Framework
Return value:	A string containing a text description of the Framework

4.9.2.3	**byte[] getFWProperty()**
Description:	Return 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.
Return value:	A memory buffer containing the Framework property

4.9.2.4	**UUID getFWPropertyID()**
Description:	Return a UUID of the format of the following Framework property
Return value:	UUID of the format of the following Framework property

4.9.2.5	**byte[] getPath()**
Description:	Return a pointer to a 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 10646:2014, Annex D). NOTE When FrameworkSchema 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.
Return value:	A pointer to a string containing the path of the file containing the Framework executable code, including the filename.

4.9.2.6	**String getProductVersion()**
Description:	Return the version string of the Framework software
Return value:	The version string of the Framework software

4.9.2.7	**String getSpecVersion()**
Description:	Return the major/minor version number of the BioAPI specification to which the Framework was implemented.
Return value:	Major/minor version number of the BioAPI specification to which the Framework was implemented.

4.9.2.8	**String getVendor()**
Description:	Return a string containing the name of the Framework vendor.
Return value:	A string containing the name of the Framework vendor.

4.10 Class GUIBitmap

4.10.1 Description

This class provides the support for exchanging graphical information between the application and the BSP, through GUI callback functions.

4.10.2 Method summary

4.10.2.1	**BIRSubtype getBIRSubtype()**
Description:	Return the subtype of the BIR
Return value:	The subtype of the BIR

4.10.2.2	**int getHeight()**
Description:	Return the height of the BIR image
Return value:	The height of the BIR image

4.10.2.3	**byte[][] getPixel()**
Description:	Return the BIR image array from left to right and up to down
Return value:	The BIR image array

4.10.2.4	**int getWidth()**
Description:	Return the width of the BIR image
Return value:	The width of the BIR image

4.11 Class IdentifyPopulation

4.11.1 Description

Represents the collection of BIRs the compare takes place against on biometric identification. The interface provides a single property, which is the list of members of the population to be used. This property is not allowed to be changed outside of this interface.

4.11.2 Method summary

4.11.2.1	**void addMember(PopulationMember member)**
Description:	Add a new member to the population to be used for identification purposes. Successive calls to this method fills in the List of the population members.
Parameters:	*member:* the member to be added.
Exception:	If input parameters are invalid or operation fails due to an error. BioAPIException (see 9.1)

4.11.2.2	**void destroy()**
Description:	Removes all the information of the list of the population used for identification.
Exception:	BioAPIException (see 9.1)

4.11.2.3	**Vector<PopulationMember> getIdentifyPopulation()**
Description:	Return the list of members of the population
Return Value:	The list of members of the population

4.11.2.4	**bool isBound()**
Description:	True if at least one BIR in the population is bound to the BSP.
Return value:	True if it is bound, False in any other case

4.11.2.5	**void unbind()**
Description:	Ensures that all member BIRs are unbound.
Exception:	If operation fails, BioAPIException (see 9.1)

4.12 Class PopulationMember

4.12.1 Description

Defines a member of the population list to be used for identification purposes.

4.12.2 Method summary

4.12.2.1	**UUID getKey()**
Description:	Return the user identifier, which is expected to be related (or being the same) as the unique identifier of the user in the database.
Return value:	The user identifier

4.12.2.2	**BIR getTemplate()**
Description:	Return the user's biometric reference
Return value:	The user's biometric reference

4.12.2.3	**void setKey(UUID key)**
Description:	Set the user identifier, which is expected to be related (or being the same) as the unique identifier of the user in the database.
Parameters:	*key:* The user identifier

4.12.2.4	**void setTemplate(BIR template)**
Description:	Set the user's biometric reference
Parameters:	*template:* the user's biometric reference

4.13 Class RegistryID

4.13.1 Description

Defines the identification of the data to be used or the product identification. It contains two parameters, as defined in the different fields of ISO/IEC 19785 BIRs.

4.13.2 Method summary

4.13.2.1	**short getOwner()**
Description:	Return the owner code
Return value:	Owner code

4.13.2.2	short getType()
Description:	Return the type code
Return value:	Type code

4.13.2.3	void setOwner(short owner)
Description:	Set the owner code
Parameters:	*owner:* Owner code

4.13.2.4	void setType(short type)
Description:	Set the type code
Parameters:	*type:* Type code

4.14 Class SecurityProfileType

4.14.1 Description

Defines the 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/valDate 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 interface UnitSchema as the output parameter of BioAPI_QueryUnits, the parameters in this structure indicate the information supported in the BioAPI_Unit. More information can be found in ISO/IEC 19784-1.

4.14.2 Method summary

4.14.2.1	Vector<BSPSchemaOptions> acBioOption()
Description:	Return a mask which indicates which security options of MAC or digital signature are supported or to be executed by the BioAPI_Unit.
Return value:	A mask which indicates which security options of MAC or digital signature are supported or to be executed by the BioAPI_Unit.

4.14.2.2	byte[] getENCInfo()
Description:	Return the ENC info
Return value:	The ENC info

4.14.2.3	byte[] getHASHAlgForACBio()
Description:	Return the HASH algorithm for ACBio
Return value:	The HASH algorithm for ACBio

4.14.2.4	byte[] getMACAlgForACBio()
Description:	Return the MAC algorithm for ACBio
Return value:	The MAC algorithm for ACBio

4.14.2.5	byte[] getMACInfo()
Description:	Return the MAC info
Return value:	The MAC info

4.14.2.6	byte[] getSIGNAlg()
Description:	Return 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 [3]) assigned to digital signature algorithms.
Return value:	The digital signature algorithm supported by a BioAPI Unit

4.14.2.7	byte[] getSIGNAlgForACBio()
Description:	Return the SIGN algorithm for ACBio
Return value:	The SIGN algorithm for ACBio

4.14.2.8	Vector<SecurityOptionsType> getSupportedSecutityOptions()
Description:	Return the supported security options
Return value:	The supported security options

4.15 Class UnitList

4.15.1 Description

Identifies a list of the selected BioAPI_Units by category and ID, where only one single unit per category is allowed. If an existing unit of a certain category is present when adding a new unit of that category, the added one will substitute the existing one.

4.15.2 Method summary

4.15.2.1	void add(UnitListElement unitListElement)
Description:	Add a new BioAPI unit to the list of selected units. If a unit of this category type is already included, the new unit replaces it.
Parameters:	*unitListElement*: element to add

4.15.2.2	int getUnitID (UnitCategoryType unitCategoryType)
Description:	Gets the unitID of the unit which has the selected category type.
Parameters:	*unitCategoryType*: category of the unit ID to look for
Return value:	Unit ID corresponding to the category type provided.
Exception:	If there is no unit of the selected category type, BioAPIException of type BioAPI-ErrUnitCategoryNotFound is thrown.

4.16 Class UnitListElement

4.16.1 Description

Identifies a BioAPI_Unit by category and ID.

4.16.2 Method summary

4.16.2.1	UnitCategoryType getUnitCategory()
Description:	Return the category of the unit
Return value:	The category of the unit

4.16.2.2	**int getUnitID()**
Description:	Return the ID assigned to the BioAPI_Unit
Return value:	The ID assigned to the BioAPI_Unit

4.16.2.3	**void setUnitCategory(UnitCategoryType unitCategory)**
Description:	Set the category of the unit
Parameters:	*unitCategory:* category of the unit

4.16.2.4	**void setUnitID(int unitID)**
Description:	Set the ID assigned to the BioAPI_Unit
Parameters:	*unitID:* ID assigned to the BioAPI_Unit

4.17 Class UnitSchema

4.17.1 Description

Defines the properties of the biometric unit.

4.17.2 Method summary

4.17.2.1	**UUID getBSPUUID()**
Description:	Return the UUID of the BSP supporting this BioAPI Unit.
Return value:	UUID of the BSP supporting this BioAPI Unit.

4.17.2.2	**String getFirmwareVersion()**
Description:	Return a string containing the version of the firmware. Empty if not available.
Return value:	string containing the version of the firmware.

4.17.2.3	**String getHardwareSerialNumber()**
Description:	Return a string containing the vendor defined unique serial number of the hardware component. Empty if not available.
Return value:	string containing the vendor defined unique serial number of the hardware component.

4.17.2.4	**String getHardwareVersion()**
Description:	Return a string containing the version of the hardware. Empty if not available.
Return value:	string containing the version of the hardware.

4.17.2.5	**int getMaxBSPDbSize()**
Description:	Return the maximum size database supported by the BioAPI Unit, in case it is an Archive unit. If zero, no database exists.
Return value:	Maximum size database supported by the BioAPI Unit, in case it is an Archive unit. If zero, no database exists.

4.17.2.6	**int getMaxIdentify()**
Description:	Return the largest identify population supported by the BioAPI Unit, in case it is a Comparison unit. Unlimited = FFFFFFFF.
Return value:	Largest identify population supported by the BioAPI Unit.

4.17.2.7	Vector<SecurityProfileType> getSecurityProfile()
Description:	Return security profile of the BioAPI Unit
Return value:	Security profile of the BioAPI Unit

4.17.2.8	String getSoftwareVersion()
Description:	Return a string containing the version of the software. Empty if not available.
Return value:	string containing the version of the software.

4.17.2.9	Vector<EventKind> getSupportedEvents()
Description:	Return a mask indicating which types of events are supported by the hardware.
Return value:	Mask indicating which types of events are supported by the hardware.

4.17.2.10	UnitCategoryType getUnitCategory()
Description:	Return the category of the BioAPI Unit
Return value:	Category of the BioAPI Unit

4.17.2.11	int getUnitID()
Description:	Return the ID of the BioAPI Unit. This will be created by the BSP uniquely.
Return value:	ID of the BioAPI Unit

4.17.2.12	UUID getUnitManagerUUID()
Description:	Return the UUID of the software component directly managing the BioAPI Unit (may be euther the BSP itself or a BFP).
Return value:	UUID of the software component directly managing the BioAPI Unit.

4.17.2.13	UUID getUnitProperties()
Description:	Return a 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.
Return value:	UUID indicating a set of properties of the BioAPI Unit.

4.17.2.14	byte[] getUnitProperty()
Description:	Return 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.
Return value:	Memory buffer containing the Unit property describing the BioAPI Unit.

4.17.2.15	UUID getUnitPropertyID()
Description:	Return the UUID of the format of the following Unit property structure.
return value:	UUID of the format of the following Unit property structure.

4.17.2.16	String getVendorInformation()
Description:	Return vendor proprietary information
return value:	String with vendor proprietary information

4.17.2.17	boolean isAuthenticatedHardware()
Description:	Return a boolean value that indicates whether the hardware component has been authenticated.
return value:	Boolean value that indicates whether the hardware component has been authenticated.

4.17.2.18	void setBSPUUID(UUID bspUUID)
Description:	Set the UUID of the BSP supporting this BioAPI Unit
Parameters:	*bspUUID:* UUID of the BSP supporting this BioAPI Unit

4.17.2.19	void setUnitID(int unitID)
Description:	Set the ID of the BioAPI Unit
Parameters:	*unitID:* ID of the BioAPI Unit

4.18 Class UUID

4.18.1 Description

Throughout the use of this part of ISO/IEC 30106, there is the need of using Unique Identifiers, either for components or for users. This class defines the Universally Unique IDentifier (UUID), coded in the java.util package.

5 Object oriented interfaces for supporting BioAPI_Units

5.1 General

Each unit should have a UnitSchema implemented to be used by the BFP and/or BSP. Also, each unit should have an ACBioInstance implemented. If ACBio is supported by this Unit, then the Unit shall update this field with the last generated ACBio instance. If ACBio is not supported, this field shall be fixed to NULL.

A comparison unit should have the following fields implemented, used internally by the Verify methods: int FMRAchieved, which indicates the value of the FMR achieved within the comparison; BIR AdaptedBIR, which, if indicated by the VerifyResultOptions, holds the resulting BIR of the template, once adapted by the result of the comparison made (e.g. for those cases with dynamic adaptation of the user's biometric reference); and byte[] AdditionalData which hold, if required, the additionalData generated by the Verify method called previously.

And, finally, in a sensor unit, the following fields should be implemented: byte[] AuxiliaryData, and UnitIndicatorStatus IndicatorStatus.

5.2 Interface archive

5.2.1 Description

This interface represents archiving functionality to a biometric application or BSP. Specific implementation of the archiving system is up to the developer (e.g. directory with files, SQL based database engine, etc.), as far as the interface follows the specification of this part of ISO 30106.

Methods and properties from this Interface shall not be accessible by the application or the framework, as this interface is only intended to be included in a BSP or a BFP.

5.2.2 Method summary

5.2.2.1	void closeDatabase (int unitID)
Description:	Closes the access to the database for which the Unit is developed.
Parameters:	*unitID*: ID of the BioAPI_Unit to perform the operation.
Exception:	If operation fails, BioAPIException (see 9.1).

5.2.2.2	void deleteBIR (int unitID, UUID key)
Description:	Deletes the BIR from database.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *key*: UUID of the record to be deleted.
Exception:	If the database has been closed, or if database was opened in read-only mode, or any other kind of error, BioAPIException (see 9.1).

5.2.2.3	BIR getSingleBIR (int unitID, UUID key, Vector<ResultOptions> resultOptions)
Description:	Get the specified record.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *key*: UUID of the record to retrieve. — *resultOptions*: requests additional output such as adapted BIR and/or the additionalData.
return value:	The requested record.
Exception:	If the database has been closed, or the database was opened in write-only mode, or the record is not found, or any other kind of error, BioAPIException (see 9.1).

5.2.2.4	Vector<UUID> listUUIDs (int unitID)
Description:	Returns the list of UUIDs included into the opened database.
Parameters:	*unitID*: ID of the BioAPI_Unit to perform the operation.
return value:	The list of UUIDs included into the opened database.
Exception:	If the database has been closed or the database was opened in write-only mode or any other kind of error, BioAPIException (see 9.1).

5.2.2.5	void openDatabase (int unitID, byte[] databaseID, BIRDatabaseAccess access)
Description:	Opens the database managed by the BSP or BFP. No other database shall be opened at the same time within the same unit. If a different database has been previously used, then the CloseDatabase method shall be called before calling OpenDatabase.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *databaseID*: specifies an optional identifier for the database to be opened. When different databases can be used with a single Archive unit, all databases shall be compatible with the internal procedures of the Archive unit. If the unit does not allow the selection of different databases, then this parameter shall have a NULL value. — *access*: specifies the access mode of the database to be opened (read/write).
return value:	The object representing the database opened.
Exception:	If the database is already opened, or any other kind of error, BioAPIException (see 9.1)

5.2.2.6	UUID storeBIR (int unitID, BIR biometricReference)
Description:	Add the specified BIR to database with no UUID, allowing the Unit to return the UUID assigned.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *biometricReference*: specific BIR to store.
return value:	The ID of the newly added BIR.
Exception:	If the database has been closed or the database was opened in read-only mod, or any other kind of error, BioAPIException (see 9.1).

5.2.2.7	UUID storeBIR (int unitID, BIR biometricReference, byte[] auxiliaryData)
Description:	Add the specified BIR to database with no UUID, allowing the Unit to return the UUID assigned. A set of bytes with additional information is submitted in AuxiliaryData. The format of that data is to be understood by the Unit (not specified in this International Standard). This information can be, for example, the demographics associated to the BiometricReference.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *biometricReference*: specific BIR to store. — *auxiliaryData*: additional data to be stored into the database in reference to the BIR stored.
return value:	The ID of the newly added BIR.
Exception:	If the database has been closed or the database was opened in read-only mode or any other kind of error, BioAPIException (see 9.1).

5.2.2.8	void storeBIR (int unitID, BIR biometricReference, UUID key)
Description:	Add the specified BIR to database and assign it the UUID provided. If the UUID is already assigned, throw an exception. If the program wanted to update an existing UUID, the application should first delete it and then re-use the UUID. This is done this way to avoid involuntary overwriting.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *biometricReference*: specific BIR to store. — *UUID*: assigned UUID to that BIR.
Exception:	If the database has been closed or the database was opened in read-only mode or the UUID is already in use, or any other kind of error, BioAPIException (see 9.1).

5.2.2.9	**void storeBIR (int unitID, BIR biometricReference, byte[] auxiliaryData, UUID key)**
Description:	Add the specified BIR to database and assign it the UUID provided. If the UUID is already assigned, throw an exception. If the program wanted to update an existing UUID, the application should first delete it and then re-use the UUID. This is done this way to avoid involuntary overwriting. A set of bytes with additional information is submitted in AuxiliaryData. The format of that data is to be understood by the Unit (not specified in this International Standard). This information can be, for example, the demographics associated to the BiometricReference.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation — *biometricReference*: specific BIR to store — *auxiliaryData*: additional data to be stored into the database in reference to the BIR stored. — *UUID*: assigned UUID to that BIR.
Exception:	If the database has been closed or the database was opened in read-only mode or the UUID is already in use, or any other kind of error, BioAPIException (see 9.1).

5.2.2.10	**IdentifyPopulation newIdentifyPopulation (int unitID)**
Description:	The database used by the BSP or BFP is set to be the data source for an identification operation. This can also be used to get a list of the BIRs included in the database.
Parameters:	*unitID:* ID of the BioAPI_Unit to perform the operation.
Return Value:	The object representing the population (collection of BIRs) (see 4.11 and 5.3).
Exception:	If the database has been closed or the database was opened in write-only mode or any other kind of error, BioAPIException (see 9.1).

5.2.2.11	**IdentifyPopulation newIdentifyPopulation (int unitID, byte[] query)**
Description:	The database used by the BSP or BFP is set to be the data source for an identification operation. This can also be called to obtain the list of BIRs that match specified criteria in the database.
Parameter:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *query*:query submitted to the database to select the users to be included into the population from which to identify. The query is submitted as a byte array, but following the specific format demanded by the database engine used.
Return value:	The object representing the population (collection of BIRs) (see 4.11 and 5.3).
Exception:	If the database has been closed or the database was opened in write-only mode or no records have been found or any other kind of error, BioAPIException (see 9.1).

5.2.2.12	**IdentifyPopulation newIdentifyPopulation (int unitID, Vector<UUID> uuidList)**
Description:	The database used by the BSP or BFP is set to be the data source for an identification operation. This can also be used to get a list of the BIRs included in the database.
Parameters:	— *unitID:* ID of the BioAPI_Unit to perform the operation. — *uuidList*: list of UUIDs to be included into the new identify population.
Return value:	The object representing the population (collection of BIRs) (see 4.11 and 5.3).
Exception:	If the database has been closed or the database was opened in write-only mode or any other kind of error, BioAPIException (see 9.1).

5.3 Interface comparison

5.3.1 Description

This interface includes all properties and methods needed to perform biometric comparison functionality. Some properties are defined to return the results of the Verify methods, securing their values as no modification is allowed publicly. Methods are overloaded to allow the comparison of single samples or to adapt the comparison algorithm internally by the use of a list of BIRs in addition to the sample BIR to be compared.

Within this unit, the term FMR is used. FMR represents the False Match Rate as a 32-bit integer value (N) that indicates a probable False Matching 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. For security reasons, no methods to modify the existing FMR object are provided.

Methods and properties from this Interface shall not be accessible by the application or the framework, as this interface is only intended to be included in a BSP or a BFP.

5.3.2 Method summary

5.3.2.1	**Vector<Candidate> identify (int unitID, int maxFMRrequested, BIR processedBIR, boolean binning, int maxResults, int timeout)**
5.3.2.2	**Vector<Candidate> identify (int unitID, int maxFMRrequested, BIR processedBIR, Vector<BIR> auxiliaryBIRs, boolean binning, int maxResults, int timeout)**
Description:	Performs biometric identification of the existing biometric sample. This function performs an identification (one-to-many) comparison between a processed BIR and a set of reference BIRs. The input is the processed BIR captured specifically for this identification. The population that the comparison takes place against shall be previously stated by calling PresetIdentifyPopulation. An overloaded method is defined as to allow using also a list of auxiliary BIRs in order to better adapt the comparison algorithm.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *maxFMRrequested*: the requested FMR criterion for successful identification (i.e. the comparison threshold). — *processedBIR*: the BIR to be identified. — *auxiliaryBIRs*: (optional) the list of auxiliary BIRs used to improve the performance of the comparison algorithm. — *binning*: a boolean indicating whether binning is on or off. Binning is a search optimization technique that the BSP may employ. It is based on searching a subset of the population according to the intrinsic characteristics of the biometric data. While it can improve the speed of the compare 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). — *maxResults*: a value of zero is a request for all candidates. — *timeout*: an integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, an exception is thrown. This value can be any positive number. A - 1 value means the BSPs default timeout value will be used.
Return value:	The list of Candidates that represents the result of the biometric operation.
Exception:	If input parameters are invalid or operation fails due to an error. BioAPIException (see 9.1).

5.3.2.3	**void presetIdentifyPopulation (int unitID, IdentifyPopulation population)**
Description:	Provides the population of BIRs to the comparison unit. After this method is invoked successfully, a BSP can call identify(). The BSP keeps this setting in effect until the presetIdentifyPopulation() is invoked with different setting or either the BSP or the session is terminated.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *population*: the population of BIRs against which the identification is performed.
Exception:	If input parameters are invalid or operation fails due to an error. BioAPIException (see 9.1).

5.3.2.4 5.3.2.5	**boolean verify (int unitID, int maxFMRrequested, BIR processedBIR, BIR referenceTemplate, Vector<ResultOptions> options)** **boolean verify (int unitID, int maxFMRrequested, BIR processedBIR, BIR referenceTemplate, Vector<BIR> auxiliaryBIRs, Vector<ResultOptions> options)**
Description:	Performs biometric verification of an existing biometric sample. This function performs a verification (one-to-one) comparison between two BIRs: the input BIR and the biometric reference. The input BIR is the processed BIR constructed specifically for this verification. The reference template was created at enrolment. The application shall request a maximum FMR value criterion (threshold) for a successful comparison. The Boolean output of the method indicates whether the verification was successful or not, and the internal properties (fmrAchieved, adaptedBIR and additionalData) are updated. By setting the RequestAdaptedBir option, the application can request that a BIR be constructed by adapting the reference template using the input processed BIR. If the comparison is successful, an attempt may be made to adapt the reference template with information taken from the input BIR (not all BSPs perform adaptation). The resulting adapted BIR should now be considered an optimal enrolment template (it is up to the application whether it uses or discards this data.). It is important to note that adaptation might not occur in all cases. If additionalData is associated with the reference template, the additionalData may be returned upon successful verification if the achieved FMR is sufficiently stringent; this is controlled by the policy of the BSP and specified in its schema. The return of additionalData is requested by setting the requestAdditionalData option. Access to the different ways of the results is provided by the Get functions of this interface. This is done in this way so that the results cannot be modified outside of the Comparison Unit. This method is overloaded with an additional parameter stating a list of auxiliary BIRs that may help to optimize the behaviour of the comparison algorithm.

Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *maxFMRrequested*: the requested FMR criterion for successful verification (i.e. the comparison threshold). — *processedBIR*: the BIR to be identified. — *auxiliaryBIRs*: (optional) the list of auxiliary BIRs used to improve the performance of the comparison algorithm. — *referenceTemplate*: the BIR to be verified against. — *options*: requests additional output such as adapted BIR and/or the additionalData.
Return value:	The boolean decision that represents the result of the biometric operation.
Exception:	If input parameters are invalid or operation fails due to an error. BioAPIException (see 9.1)

5.3.2.6	**BIR getAdaptedBIR(int unitID)**
Also, as mentioned, this interface have the following properties, used internally by the Verify methods.	
Description:	Return the resulting BIR of the template, once adapted by the result of the comparison made (e.g. for those cases with dynamic adaptation of the user's biometric reference).
Parameters:	*unitID*: ID of the BioAPI_Unit to perform the operation.
Return value:	Resulting BIR of the template

5.3.2.7	**int getFMRAchieved(int unitID)**
Description:	Return the value of the FMR achieved within the comparison
Parameters:	*unitID*: ID of the BioAPI_Unit to perform the operation
Return value:	Value of the FMR achieved

5.4 Interface processing

5.4.1 Description

Represents the group of biometric operation that are available when the unit of processing category is being used.

Methods and properties from this Interface shall not be accessible by the application or the framework, as this interface is only intended to be included in a BSP or a BFP.

5.4.2 Method summary

5.4.2.1	**BIR createTemplate (int unitID, BIR capturedBIR, BIR referenceTemplate, RegistryID outputFormat, byte[] additionalData)**
5.4.2.2	**BIR createTemplate (int unitID, Vector<BIR> capturedBIRs, BIR referenceTemplate, RegistryID outputFormat, byte[] additionalData)**
5.4.2.3	**BIR createTemplate (int unitID, BIR capturedBIR, BIR referenceTemplate, Vector<BIR> auxBIRs, RegistryID outputFormat, byte[] additionalData)**
5.4.2.4	**BIR createTemplate (int unitID, Vector<BIR> capturedBIRs, BIR referenceTemplate, Vector<BIR> auxBIRs, RegistryID outputFormat, byte[] additionalData)**

Description:	Takes a BIR or a list of BIRs containing biometric data for the purpose of creating a new enrolment template. A new BIR is constructed from the captured BIR and it may perform an update based on an existing reference template. The optional input reference template is provided for use in creating a new template if the BSP supports this capability.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *capturedBIR*: the captured BIR. — *capturedBIRs*: the list of captured BIRs. — *referenceTemplate*: optionally, the existing template to be updated. — *auxBIRs*: optionally, list of auxiliary BIRs. This exta BIRs maybe used to provide feedback to the tamplate creation operation or modify it based on statistical operation. — *outputFormat*: specifies which BDB format to use for the returned processed BIR, if the BSP supports more than one format. A null value indicates that the BSP is to select the format. — *additionalData*: a additionalData that will be stored by the BSP.
Return value:	Returns the BIR object that represents the result of processing.
Exception:	If the input parameters are invalid, the format is not supported or operation fails due to error. BioAPIException (see 9.1)

5.4.2.5	**BIR process (int unitID, BIR capturedBIR, RegistryID outputFormat)**
5.4.2.6	**BIR process (int unitID, BIR captureBIR, Vector<BIR> auxiliaryBIRs, RegistryID outputFormat)**
Description:	Processes the biometric sample captured, in order to create a processed biometric sample for the purpose of either verification or identification. The overloaded method enables implementations that require the input of auxiliary data to process the operation.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *capturedBIR*: the captured BIR. — *auxiliaryBIR*: a BIR containing the auxiliary data used in the operation. — *outputFormat*: specifies which BDB format to use for the returned processed BIR, if the BSP supports more than one format. Null indicates that the BSP is to select the format.
Return value:	The BIR object tht represents the result of processing.
Exception:	If the input parameters are invalid or operation fails due to error. BioAPIException (see 9.1)

5.5 Interface sensor

5.5.1 Description

Represents the group of biometric operations that are available in a sensor unit, which is in charge of acquiring the sample.

Methods and properties from this Interface shall not be accessible by the application or the framework, as this interface is only intended to be included in a BSP or a BFP.

5.5.2 Method summary

5.5.2.1	**void calibrate (int unitID, int timeout)**
Description:	Performs a calibration of the sensor if the sensor supports it
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation — *timeout*: The integer value specifying the timeout value in milliseconds. -1 means the BSPs default value will be used
Exception:	If the input parameters are invalid or operation fails due to error. BioAPIException (see 9.1)

5.5.2.2	**BIR capture (int unitID, Vector<Purpose> purpose, BiometricSubtype biometricSubtype, RegistryID outputFormat, int timeout, Vector<ResultOptions> resultOptions)**
Description:	Captures samples for the purpose specified and the BSP returns either an intermediate type BIR or a processed BIR. The purpose is recorded in the header of the captured BIR. If RequestAuditData option is specified, a BIR of type raw may be returned in CaptureResult. The BSP is responsible for serializing.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *purpose*: indicates the purpose of the biometric data capture. — *biometricSubtype*: indicates the subtype of the biometric sample acquired. Null value indicates value is not provided. — *outputFormat*: specifies which BDB format to use for the returned processed BIR, if the BSP supports more than one format. A null value indicates that the BSP is to select the format. — *timeout*: integer value specifying the timeout value in milliseconds. — *resultOptions*: requests additional output such as additional data.
Return value:	The BIR object that represents the result of the capture operation.
Exception:	If the sensor device is busy or operation failed. BioAPIException (see 9.1).

5.5.2.3	**UnitIndicatorStatus getIndicatorStatus(int unitID)**
Description:	Return the unit indicator status.
Parameters:	*unitID*: ID of the BioAPI_Unit to perform the operation.
Return value:	The indicator status.

5.5.2.4	**void setIndicatorStatus (int unitID, UnitIndicatorStatus indicatorStatus)**
Description:	This function sets the selected BioAPI Unit to the requested indicator status if the BioAPI Unit supports it. After Accept or Reject is set in the IndicatorStatus parameter, the status will not be changed until the application sets another value.
Parameters:	— *unitID*: ID of the BioAPI_Unit to perform the operation. — *indicatorStatus*: A value to which to set the indicator status of the BioAPI_Unit.
Exception:	If case of any error, BioAPIException (see 9.1).

6 BFP level

6.1 Interface BFP

6.1.1 Description

Represents the Biometric Function Provider. This interface is composed of the BFP functionality and the integration of those BioAPI_Units supported by the BFP. Only one category of BioAPI_Units shall be included within the same BFP. It may also provide support to the communication with BSPs and the component registry.

6.1.2 Imported interfaces

One of the following interfaces shall be imported by BFP, as many times as BioAPI_Units are whenever a corresponding BioAPI_Unit is supported by the BSP:

— archive;

— sensor;

— processing;

— comparison.

6.1.3 Method summary

In addition to the methods for the categoriy selected for the BSP (given in 5.2, 5.3, 5.4 or 5.5), the following methods shall be included in BFP.

6.1.3.1	**void bfpLoad (BFPEventListener bfpEventListener, BFPGUIProgressListener bfpGUIProgressListener)**
Description:	Initializes a BFP. Initialization includes registering the BSP event handler for the specified BFP and enabling all events. The BSP can choose to provide an event handler function to receive notification of events. Many BSPs can independently and concurrently load the same BFP and each BSP can establish its own event handler. They will all receive notification of an event. The same or different event handlers can be used if a BSP loads multiple BFPs. A BSP may establish as many event handlers as it wishes for a given BFP by calling BFPLoad one or more times for that BFP. An event handler is identified by the address of the notification. When an event occurs in a BFP, the BFP may send an event notification to the BSP by calling the BSP's event handler. If a BSP has set up multiple event handlers, they shall be called one at a time (in any order chosen by the BFP) rather than concurrently. Event notification may occur at any time, either during a BSP call (related or unrelated to the event) or while there is no BSP call in execution. BSP writers should ensure that all callbacks are properly and safely handled by the BSP, no matter when the BSP receives them. When a BFP is loaded (bfpLoad), it shall raise an "insert" event immediately for each present BioAPI Unit. This will indicate to the BSP that it can go ahead and use the Unit. 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. The bfpNotifyCallback defines a callback used to notify the BSP of events of type BioAPI_EVENT. The BFP shall retain this information for later use.
Parameters:	— *bfpEventListener*: defines a callback used to notify the BioAPI Framework of events in any ongoing process. — *bfpGUIProgressListener*: defines a callback used to notify the BioAPI Framework of events in any ongoing process.
Exception:	Thrown if any the parameters is not valid or any other error during initialization. BioAPIException (see 9.1)

6.1.3.2	**byte[] controlUnit (int unitID, int controlCode, byte[] inputData)**
Description:	Sends control data to the BioAPI_Unit and receives status or operation data from there. The content of the parameters and the output will be specified in the related interface specification of this BioAPI_Unit.
Parameters:	— *unitID*: identifier of the unit to which the ControlUnit function is forwarded. — *controlCode*: function code in the BioAPI_Unit to be called. — *inputData*: buffer containing the data to be sent to the BioAPI unit related to the given ControlCode.
Return value:	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 value shall be NULL.
Exception:	If case of any error, BioAPIException (see 9.1).

6.1.3.3	byte[] getACBioInstance(int unitID)
Description:	Return the ACBioInstance. If ACBio is supported by this Unit, then the Unit shall update this property with the last generated ACBio instance. If ACBio is not supported, this property shall be fixed to NULL. NOTE The ACBioInstance may also be returned as part of the BIR, within the Security Block field.
Parameters:	*unitID*: ID of that BioAPI Unit defined by a prior operation. The UnitID has been filled into the UnitId field of each element of the UnitSchema.
Return value:	The ACBioInstance

6.1.3.4	byte[] getAuxiliaryData(int unitID)
Description:	Return the auxiliary data
Parameters:	*unitID*: ID of that BioAPI Unit defined by a prior operation. The UnitID has been filled into the UnitId field of each element of the UnitSchema.
Return value:	The auxiliary data

6.1.3.5	BFPSchema getBFPSchema()
Description:	Returns the BFP Schema
Return value:	The BFP schema

6.1.3.6	Vector<UnitSchema> queryUnits ()
Description:	This function returns a list of BioAPI Unit schemas, which are managed by the given BFP and are currently in the inserted state. This function shall only be called after BFPLoad has been called for the specified BFP. All Units in a BFP shall have a UnitSchema defined. There is no requirement that a unit ID, returned by this function for a given BioAPI unit, be made available with the same unit ID value by the BSP to the framework. A BSP is free to translate any unit ID value (provided by a BFP) into a different unit ID value before providing it to the framework. The purpose of such a translation would be to avoid the existence of duplicate unit IDs within the scope of the BSP, which might happen when a BSP is using two or more BFPs of the same category, or when a BSP is using a BSFP while also directly managing biometric sensors.
Return value:	List of UnitSchemas where each element describes properties of each unit available for the current session.
Exception:	Thrown if the method is called after the BFP is no longer available. BioAPIException (see 9.1)

6.1.3.7	void setPowerMode (int unitID, UnitPowerMode powerMode)
Description:	This function sets the referenced BioAPI Unit to the requested power mode if the BioAPI Unit supports it.
Parameters:	— *unitID:* identifier of the unit to which the ControlUnit function is forwarded. — *powerMode*: the indicator of the power mode to set the BioAPI_Unit to.
Exception:	If case of any error, BioAPIException (see 9.1)

7 BSP level

7.1 Interface BSP

7.1.1 Description

Represents the Biometric Service Provider. It is the factory of the session objects that provide access to biometric operations. This interface is composed of the BSP functionality and the integration of those BioAPI_Units supported by the BSP. It may also provide support to the communication with BFPs and the component registry.

The developer of the BSP may decide to not support the publication of certain functionality of the supported BioAPI_Units, providing the call of those methods to the external world, but launching a BioAPI Exception indicating that such a method is not supported.

EXAMPLE A BSP, for security reasons, might only want to publish aggregated functionality that can be called atomically (e.g. Enrol), not allowing access to the individual methods used for this functionality, such as Capture or CreateTemplate.

In addition, those methods and properties needed for interfacing with the framework are added by inheriting the BSPUnitSet interface.

7.1.2 Imported interfaces

The following interfaces shall be imported by BSP, whenever a corresponding BioAPI_Unit is supported by the BSP:

— archive;

— sensor;

— processing;

— comparison.

7.1.3 Method summary

In addition to the methods for all the categories of BioAPI_Units that have been given in 5.2, 5.3, 5.4 and 5.5, the following methods shall be included in BSP, although the developer may decide that a certain number of them are not supported, returning when they are called the corresponding BioAPIException (see 9.1)

7.1.3.1	**void bspLoad (BSPEventListener bspEventListener, BFPEventListener bfpEventListener, BFPGUIProgressListener bfpGUIProgressListener, BFPEnumerationListener bfpEnumerationListener)**
Description:	Initialize the BSP. It shall not be called more than once without the corresponding call to bspUnload().
Parameters:	— *bspEventListener*: defines a callback to subscribe events related to BSP. — *bfpEventListener*: defines a callback used to notify the BioAPI Framework of events in any ongoing process. — *bfpGUIProgressListener*: defines a callback used to notify the BioAPI Framework of events in any ongoing process. — *bfpEnumerationListener*: defines a callback used to identify the BFPs installed in the framework.
Exception:	Thrown if any the parameters are not valid or any other error during initialization. BioAPIException (see 9.1)

7.1.3.2	**void bspUnload ()**
Description:	Disables events and de-registers the use of the current BSP in the application.
Exception:	Thrown if any the parameters is not valid or any other error during initialization. BioAPIException (see 9.1)

7.1.3.3	**byte checkQuality (BIR inputBIR, RegistryID qualityAlgorithmID)**
Description:	This function performs a quality assessment of the biometric data contained within an input BIR. If a quality algorithm is specified and that algorithm is supported by the BSP, it shall be utilized. If NULL, the BSP will select the quality algorithm to apply. BSPs may determine what quality algorithms are supported by calling BioAPI_EnumBSPs. If an unsupported algorithm is requested, a BioAPIERR_UNSUPPORTED_ALGORITHM BioAPIException shall be thrown.
Parameters:	— *inputBIR*: BIR containing the biometric data whose quality is to be assessed. — *qualityAlgorithm*: as input, the quality algorithm to be used by the BSP; as output, the quality algorithm actually used by the BSP.
Return value:	The assessed quality of the BIR data.
Exception:	If case of any error, BioAPIException (see 9.1)

7.1.3.4	**byte[] controlUnit (int unitID, int controlCode, byte[] inputData)**
Description:	Sends control data to the BioAPI_Unit and receives status or operation data from there. The content of the parameters and the output will be specified in the related interface specification of this BioAPI_Unit.
Parameters:	— *unitID*: identifier of the unit to which the ControlUnit function is forwarded. — *controlCode*: the function code in the BioAPI_Unit to be called. — *inputData*: buffer containing the data to be sent to the BioAPI unit related to the given ControlCode.
Return value:	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 value shall be NULL.
Exception:	If case of any error, BioAPIException (see 9.1)

7.1.3.5	**UUID enrol (UnitList unitList, int numberOfPresentations, int numberOfAttempts, int numberOfTransactions, int qualityThreshold, int maxFmrRequested, int timeout, Vector<Purpose> purposes, BiometricSubtype biometricSubtype, RegistryID outputFormat, byte[] additionalData)**
7.1.3.6	**UUID enrol (UnitList unitList, int numberOfPresentations, int numberOfAttempts, int numberOfTransactions, int qualityThreshold, int maxFmrRequested, int maxFmrRequestedForUpdate, UUID referenceId, int timeout, Vector<Purpose> purposes, BiometricSubtype biometricSubtype, RegistryID outputFormat, byte[] additionalData)**
7.1.3.7	**UUID enrol (UnitList unitList, Vector<BIR> capturedBirs, int maxFmrRequested, int timeout, Vector<Purpose> purposes, BiometricSubtype biometricSubtype, RegistryID outputFormat, byte[] additionalData)**
7.1.3.8	**UUID enrol (UnitList unitList, Vector<BIR> capturedBirs, int maxFmrRequested, int maxFmrRequestedForUpdate, UUID referenceId, int timeout, Vector<Purpose> purposes, BiometricSubtype biometricSubtype, RegistryID outputFormat, byte[] additionalData)**

Description:	Enrol a user by means of (in order of appereance in the list above) — providing a list of samples to use for the enrolment in the first parameter of the method, and — requiring the capture process of a certain number of times to the Sensor unit in the BSP. The enrolment can be done from scratch or by updating (replacing) a previous enrolment by using referenceId parameter. In the first case, referenceId can be used to assign a UUID to the new template. In the second case, the replacing operation should be done by using a UUID already present in the database used for enrolment. In this case, if the biometric template corresponding to the referenceId provided does match with the new template, a replacing operation should be done. Otherwise, a BioAPI.invalidUUID exception should be thrown. — The result of a successful enrolment will be the UUID assigned to the biometric reference created and, optionally, the BIR for that biometric reference. The BSP is responsible for providing the user interface associated with the enrolment operation as a default. The application may request control of the GUI "look-and-feel" by providing a GUI callback via the BSP.subscribeToGUIEvents() method. Since the enrol() operation includes capture, it serializes the 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 throwing an exception to indicate that the device is busy or by queuing requests.

Parameters:	— *(sensor/archive/processing)unitID*: units ids to use to perform the operation. — *numberOfPresentations*: minimum number of images to build a sample. — *numberOfAttempts*: maximum number of capture attempts until FTA error. — *numberOfTransactions*: when the sample to enrol is to be captured within the BSP, this parameter determine the number of samples to be acquired to the user. — *qualityThreshold*: the requested quality threshold criterion for successful capture. — *maxFmrRequested*: the requested FMR criterion for successful enrolment (i.e. the comparison threshold). — *maxFmrRequestedForUpdating*: if an updating operation needs to be performed, this parameter represents the maximum FMR value for successful re-enrolment. — *referenceId*: the UUID of the biometric reference to be assigned or updated. — *timeout*: 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. — *capturedBIRs:* a list of BIRs with biometric samples to be used for the enrolment. — *purpose*: a value indicating the desired purpose (Enrol, EnrolForVerificationOnly, or EnrolForIdentification only). — *biometricSubtype*: specifies which subtype (e.g., left/right eye) to enrol. A 'null' value indicates that the BSP is to select the subtype(s). — *outputFormat*: specifies which BDB format to use for the returned NewTemplate, if the BSP supports more than one format. A NULL value indicates that the BSP is to select the format — *additionalData*: additionalData that will be stored by the BSP. — *resultOptions*: requests additional output such as audit data. This data may be used to provide a human-identifiable data of the person at the sensor unit.
Return value:	The UUID for the biometric reference stored.
Exception:	If any of the parameters is not correct or other error occur during the process (e.g. no access to the database), BioAPIException (see 9.1).

7.1.3.9	**byte[] getACBioInstance(int unitId)**
Description:	Return the ACBio info
Parameters:	*unitID*: ID of the BioAPI Unit.
Return value:	The ACBio data

7.1.3.10	**byte[] getAdditionalData(int unitID)**
Description:	Return the auxiliary data
Parameters:	*unitID*: ID of that BioAPI Unit defined by a prior operation. The UnitID has been filled into the UnitId field of each element of the UnitSchema.
Return value:	The auxiliary data

7.1.3.11	**BSPSchema getBSPSchema()**
Description:	Returns the BSP Schema
Return value:	The BSP schema

7.1.3.12	**Vector<Candidate> identifyAggregated (UnitList unitList, int maxFMRrequested, BiometricSubtype biometricSubtype, boolean binning, int maxResults, int timeout, Vector<ResultOptions> options)**
7.1.3.13	**Vector<Candidate> identifyAggregated (UnitList unitList, BIR inputBIR, int maxFMRrequested, BiometricSubtype biometricSubtype, boolean binning, int maxResults, int timeout, Vector<ResultOptions>**
7.1.3.14	**Vector<Candidate> identifyAggregated (UnitList unitList, Vector<BIR> inputBIRs, int maxFMRrequested, BiometricSubtype biometricSubtype, boolean binning, int maxResults, int timeout, Vector<ResultOptions> options)**
Description:	This method provides an aggregated functionality. It increases the functionality of the Identify method from Comparison (see 5.3), allowing the Identify operation either, when the biometric sample is directly captured by the BSP calling its own Sensor unit, or by providing the input BIR in raw or processed format. A call to PresetIdentifyPopulation shall be done before this method can be called in order to establish the population within which the search is done. Refer to the Identify methods in Comparison (5.3) for further explanation. The BSP is responsible for providing the user interface associated with the enrolment operation as a default. The application may request control of the GUI "look-and-feel" by providing a GUI callback via the BSP.subscribeToGUIEvents() method. Since this operation includes capture, it serializes the 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 throwing an exception to indicate that the device is busy or by queuing requests.
Parameters:	— *(sensor/archive/processing/comparison)unitID*: units ids to use to perform the operation. — *inputBIR*: BIRs to perform the identification, which could be in raw or processed format. — *inputBIRs*: list of captured BIRs to perform the identification, which could be in raw or processed format. — *maxFMRRequested*: FMR threshold — *biometricSubtype*: specifies which subtype (e.g. left/right eye) to capture. Null indicates that the value is not provided. — *binning*: a bool indicating whether binning is on or off. Binning is a search optimization technique that the BSP may employ. It is based on searching a subset of the population according to the intrinsic characteristics of the biometric data. While it c improve the speed of the compare 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). — *maxResults*: value of zero is a request for all candidates. — *timeout*: integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, an exception is thrown. This value can be any positive number. -1 value means the BSPs default timeout value will be used. — *options*: requests additional output such as audit data. For the rest of the parameters, see the Identify methods in Comparison (5.3).
Return value:	See the Identify methods in Comparison (5.3).
Exception:	In case of any of the Capture exceptions (5.5) or the Identify exceptions (5.3), BioAPIException (see 9.1).

7.1.3.15 7.1.3.16	**Vector<BFPListElement> queryBFPs ()** **Vector<BFPListElement> queryBFPs (Vector<UnitCategoryType> unitCategories)**
Description:	Returns the identification of those BFPs available that are supported by the BSP in the current session. NOTE On an incoming call to this function, the BSP can use the BFP Enumeration Handler callback (see 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.
Parameters:	*unitCategories*: optionally, the list of categories for which the enumeration of the unit schemas is requested.
Return value:	List of BFPs that are supported by current BSP.
Exception:	Thrown if the method is called after the BSP is no longer available or any other error. BioAPIException (see 9.1)

7.1.3.17 7.1.3.18	**Vector<UnitSchema> queryUnits ()** **Vector<UnitSchema> queryUnits (Vector<UnitCategoryType> unitCategories)**
Description:	Returns Units available by the BSP in the current session. All Units in a BFP shall have a UnitSchema defined.
Parameters:	*unitCategories:* optionally, the list of categories for which the enumeration of the unit schemas is requested.
Return value:	List of UnitSchemas where each element describes properties of each unit available for the current session.
Exception:	Thrown if the method is called after the BSP is no longer available. BioAPIException (see 9.1).

7.1.3.19	**void setPowerMode (int unitID, UnitPowerMode powerMode)**
Description:	This function sets the referenced BioAPI Unit to the requested power mode if the BioAPI Unit supports it.
Parameters:	— *unitID:* identifier of the unit to which the ControlUnit function is forwarded. — *powerMode*: indicator of the power mode to set the BioAPI_Unit to.
Exception:	In case of any error, BioAPIException (see 9.1).

7.1.3.20	**void subscribeToGUIEvents (GUISelectEventListener guiSelectEventListener, GUIStateEventListener guiStateEventListener, GUIProgressEventListener guiProgressEventListener)**
Description:	This method provides to the BSP the callback functions for the Select, State and Progress events. If a certain event is not supported, a NULL value shall be provided for that kind of event. It is not possible to call this method with the three events set to NULL. If this method is called with a determined event that has been previously assigned a callback function, the method simply replaces the old callback address with the one provided in the current call.
Parameters:	— guiSelectEventListener: specifies address to the callback function for the Select Event. — guiStateEventListener: specifies address to the callback function for the State Event. — guiProgressEventListener: specifies address to the callback function for the Progress Event.
Exception:	BioAPIException (see 9.1)

7.1.3.21	**void unsubscribeFromGUIEvents ()**
Description:	This method clears the callback addresses previously subscribed. After a call to this function, the BSP shall cease to notify GUI events to the framework or the application.
Exception:	BioAPIException (see 9.1)

7.1.3.22	**boolean verifyAggregated (UnitList unitList, int maxFMRrequested, BIR referenceTemplate, BiometricSubtype subtype, int timeout, Vector<ResultOptions> options)**
7.1.3.23	**boolean verifyAggregated (UnitList unitList, int maxFMRrequested, UUID referenceKey, BiometricSubtype subtype, int timeout, Vector<ResultOptions> options)**
7.1.3.24	**boolean verifyAggregated (UnitList unitList, int maxFMRrequested, BIR inputBIR, BIR referenceTemplate, BiometricSubtype subtype, int Timeout, Vector<ResultOptions> options)**
7.1.3.25	**boolean verifyAggregated (UnitList unitList, int maxFMRrequested, BIR inputBIR, UUID referenceKey, BiometricSubtype subtype, int timeout, Vector<ResultOptions> options)**

Description:	This method provides an aggregated functionality. It increases the functionlality of the Verify method from IComparison (see 5.3), allowing the verify operation in the following cases: — biometric sample is captured directly by the BSP calling its own Sensor unit; therefore, the inputBIR parameter is not present; — biometric sample is supplied either in raw or processed format; — biometric reference is declared by its UUID; therefore, the referenceKey parameter is added; — biometric reference is provided as a BIR; therefore, the referenceTemplate parameter is used; — biometric reference is intrinsically known by the BSP (e.g. a single-user, single enrolment application); therefore, the call will be given with NULL as the referenceTemplate or referenceKey. Refer to the Verify methods in Comparison (5.3) for further explanation. The BSP is responsible for providing the user interface associated with the enrolment operation as a default. The application may request control of the GUI "look-and-feel" by providing a GUI callback via the BSP.subscribeToGUIEvents() method. Since this operation includes capture, it serializes the 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 throwing an exception to indicate that the device is busy or by queuing requests.
Parameters:	— *unitList*: list of unitsListElements to use to perform the operation. — *inputBIR*: sample to be verified, either in raw or processed format. — *referenceKey*: UUID of the biometric reference to be used for verification. — *referenceTemplate*: BIR corresponding to the biometric reference to be used for verification. — *maxFMRRequested*: FMR threshold. — *biometricSubtype*: specifies which subtype (e.g. left/right eye) to capture. Null indicates that the value is not provided. — *timeout*: an integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, an exception is thrown. This value can be any positive number. -1 value means the BSPs default timeout value will be used. — *options*: requests additional output such as audit data. For the rest of the parameters, see the Verify methods in Comparison (5.3).
Return value:	See the Verify methods in Comparison (5.3).
Exception:	In case of any of the Capture exceptions (5.5) or the Verify exceptions (5.3), BioAPIException (see 9.1).

8 Framework level

8.1 Interface ComponentRegistry

8.1.1 Description

Represents the interface to the component registry. The Installer adds, deletes, or modifies BSP and BFP records in the component registry managed by the framework.

The way the component registry is implemented is dependent on the developer, as the components are managed from the provided methods. Further information about the information that the component registry shall contain internally (e.g. a framework, BSP, Units schemas) is available in ISO/IEC 19784-1, Clause 10.

8.1.2 Method summary

8.1.2.1	void installBFP (BFPSchema bfpSchema, boolean update)
Description:	Installs or updates the references to a BFP in the component registry. This function is handled internally within the BioAPI Framwork and is not passed through to a BFP.
Parameters:	— *bfpSchema*: specifies the information on the BFP to be installed or updated. — *update*: if true, an update of an existing BFP is performed (i.e. if such BFP is not yet installed, it will return an ERR_COMPONENT_NOT_REGISTERED BioAPIException). If false, the installation refers to a new BFP. If the BFP is already installed, an ERR_COMPONENT_ALREADY_REGISTERED BioAPIException is thrown.
Exception:	Thrown if any error during installation. BioAPIException (see 9.1)

8.1.2.2	void installBSP (BSPSchema bspSchema, boolean update)
Description:	Installs or updates the references to a BSP in the component registry. This function is handled internally within the BioAPI Framwork and is not passed through to a BSP.
Parameters:	— *bspSchema*: specifies the information on the BSP to be installed or updated. — *update*: if true, an update of an existing BSP is performed (i.e. if such BSP is not yet installed, it will return an ERR_COMPONENT_NOT_REGISTERED BioAPIException). If false, the installation refers to a new BSP. If the BSP is already installed, an ERR_COMPONENT_ALREADY_REGISTERED BioAPIException is thrown.
Exception:	Thrown if any error during installation. BioAPIException (see 9.1)

8.1.2.3	void uninstallBFP (UUID bfpUUID)
Description:	This function uninstalls a BFP by removing references to that BFP in the component registry. This function is handled internally within the BioAPI Framework and is not passed through to a BFP.
Parameters:	*bfpUUID*: specifies BFP to be uninstalled.
Exception:	Thrown if any error. BioAPIException (see 9.1)

8.1.2.4	void uninstallBSP (UUID bspUUID)
Description:	This function uninstalls a BSP by removing references to that BSP in the component registry. This function is handled internally within the BioAPI Framework and is not passed through to a BSP.
Parameters:	*bspUUID*: specifies BSP to be uninstalled.
Exception:	Thrown if any error. BioAPIException (see 9.1)

8.2 Interface framework

8.2.1 Description

Represents the biometric system. The biometric system is the hierarchical assembly whose root node is the Framework component. The Framework controls BSPs. To provide necessary services to the Framework BSPs use BFP modules that are libraries implementing biometric algorithms and code that manages sensors hardware. Along with BSPs, another part of the Framework is component registry that stores the information about BSPs and BFPs.

8.2.2 Inherited interfaces

The ComponentRegistry is inherited by IFramework as to maintain a component registry and being able to install, uninstall components, as well as to access its internal information to allow the execution of methods such as enumBFPs, queryUnits, etc.