INTERNATIONAL STANDARD

ISO/IEC 30106-3

Second edition
2020-11

Information technology — Object oriented BioAPI —

Part 3: C# implementation

Technlogies de l'information — Objet orienté BioAPI —

Partie 3: Mise en oeuvre de C#

Reference number
ISO/IEC 30106-3:2020(E)

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

Published in Switzerland

Contents

Page

Foreword

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

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) or the IEC list of patent declarations received (see http://patents.iec.ch).

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

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

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

This second edition cancels and replaces the first edition (ISO/IEC 30106-3:2016), which has been technically revised.

The main changes compared to the previous edition are as follows:

— correction of typing errors;

— addition of AnalyseQuality method.

A list of all parts in the ISO/IEC 30106 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document specifies an application programming interface expressed in C# language. C# 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 the Microsoft.NET platform.

One of the advantages of using C# is that, as it is designed for the CLI (Common Language Infrastructure), it allows multiple high-level languages to be used on different computer platforms without being rewritten for specific architectures.

C# shares some features (overloading, some syntactic details) with C++ but also includes new characteristics (reference and output parameters, enumerations, unified type system). Furthermore, C# is very similar to Java (interfaces, exceptions, object-orientation), which implies that the structure of interfaces and namespaces (which is the equivalent to packages in Java language) is mostly the same as Java but, as expected, code implementation and compilation are different.

As Java implementation allows an easy use of Java BSPs, Java-based application servers or Java applets, C# is the best way to write windows desktop and web applications/services and provides an advanced and well-designed remote framework.

Information technology — Object oriented BioAPI —

Part 3:
C# implementation

1 Scope

This document specifies an interface of a BioAPI C# framework and BioAPI C# BSP which mirror the corresponding components specified in ISO/IEC 30106-1. The semantic equivalence of this document will be maintained with ISO/IEC 30106-2 (Java implementation). In spite of the differences in actual parameters passed between functions, the names and interface structure are the same.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 10646:2017, *Information technology — Universal Coded Character Set (UCS)*

ISO/IEC 30106-1, *Information technology — Object oriented BioAPI — Part 1: Architecture*

3 Terms and definitions

No terms and definitions are listed in this document.

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

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

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

4 BioAPI C# Namespace Structure

4.1 Overall structure

The BioAPI C# interface is divided into several namespaces. The following is the namespace structure:

— Namespace BioAPI: Contains functionality to manage units, BSPs, BFPs, the Framework and Applications.

— Namespace BioAPI.Data: Contains all the data structures.

4.2 Namespace BioAPI

4.2.1 Namespace description

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

4.2.2 Structure

The description of this namespace is given explaining a bottom-up structure. In Clause 6, 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 be either the Biometric Service Provider (BSP) or the Biometric Function Provider (BFP), but the specifications in this 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 7) and BSP (Clause 8) 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 9, with the Framework Interface and the Component Registry) and the Application interaction (Clause 10, with the specification of the Exceptions and Callback functions). This provides the equivalence to the API interface in ISO/IEC 19784-1.

4.3 Namespace BioAPI.Data

4.3.1 Namespace description

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

4.3.2 Structure

Several data structures are provided to comply with the requirements of specifying this document. All the BioAPI.Data namespace are specified in Clause 5, where all needed classes and enumerations are defined. This has to be complemented to the constants defined in ISO/IEC 30106-1.

5 Data types and constants

5.1 Class ACBioParameters

5.1.1 Description

Provides the information which is used to generate ACBio instances.

5.1.2 Properties summary

— int[] Challenge {get;} – 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.

— int[] InitialBPUIOIndexOutput {get;} – 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 accepted by the BSP, and assigned to the BSP Units and BSPs.

— int[] SupremumBPUIOIndexOutput {get;} – 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.

5.2 Class BFPListElement

5.2.1 Description

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

5.2.2 Properties summary

— UnitCategoryType UnitCategory { get; set; }: The category of the unit.

— UUID BFPID { get; set; }: The UUID assigned to the BFP.

5.3 Class BFPSchema [Serializable()]

5.3.1 Description

Represents the record in the component registry that defines the properties of the BFP installed in the system. Is a serializable class.

5.3.2 Properties summary

— UUID BFPUUID {get;}: UUID of the BFP.

— UnitCategoryType BFPCategory {get;}: Category of the BFP identified by the BFPUUID.

— String BFPDescription {get;}: A NULL-terminated string containing a text description of the BFP.

— String Path {get;}: A pointer to a NULL-terminated 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 10646:2017, 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.

— String SpecVersion {get;}: Major/minor version number of the BioAPI specification to which the BFP was implemented.

— String ProductVersion {get;}: The version string of the BFP software.

— String Vendor {get;}: A NULL-terminated string containing the name of the BFP vendor.

— sbyte[] BFPProperty {get;}

— List<RegistryID> BFPSupportedFormats {get;}: A list of the data formats that are supported by the BFP (see 7.1).

— List<BiometricType> FactorsMask {get;}: A list of the biometric types supported by the BFP (see 7.1).

— UUID FwPropertyID {get;}: UUID of the format of the following BFP property.

— byte[] FwProperty {get;}: 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.

5.3.3 Method summary

5.3.3.1 virtual void Dispose ()

Description:	Removes all the information in the current object, leaving it empty for a next use.
Exception:	None.

5.4 Class BIR

5.4.1 Description

This interface represents BIRs (Biometric Information Records). It supports the ISO/IEC 19785 series 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.

5.4.2 Properties summary

NOTE The description of each of the properties can be found in ISO/IEC 19785-1.

— RegistryID SelfID { get; set; } (see 5.13).

— byte CBEFFVersion { get; set; }.

— byte PatronHeaderVersion { get; set; }.

— RegistryID BDBFormat { get; set; } (see 5.13).

— bool BDBEncription { get; set; }.

— bool BIRIntegrity { get; set; }.

— BiometricType BDBBiometricType { get; set; } (see 5.7.2.2).

— BiometricSubtype BDBBiometricSubtype { get; set; } (see 5.7.2.1).

— RegistryID BDBCaptureDevice { get; set; } (see 5.13).

— RegistryID BDBFeatureExtractionAlg { get; set; } (see 5.13).

— RegistryID BDBComparisonAlg { get; set; } (see 5.13).

— RegistryID BDBCompresionAlg { get; set; } (see 5.13).

— RegistryID BDBPADTechnique { get; set; } (see 5.13).

— byte[] BDBChallengeResponse { get; set; }.

— Date BDBCreationDate { get; set; } (see 5.8).

— byte[] BDBIndex { get; set; }.

— ProcessedLevel BDBProcessedLevel { get; set; }.

— RegistryID BDBProduct { get; set; } (see 5.13).

— Purpose BDBPurpose { get; set; }.

— byte BDBQuality { get; set; }.

— RegistryID BDBQualityAlg { get; set; } (see 5.13).

— List<Date> BDBValidityPeriod { get; set; } // 2 dates (see 5.8).

— Date BIRCreationDate { get; set; } (see 5.8).

— byte[] BIRCreator { get; set; }.

— byte[] BIRIndex { get; set; }.

— byte[] BIRPayload { get; set; }.

— byte[] BIRPointer { get; set; }.

— List<Date> BIRValidityPeriod { get; set; } // 2 dates (see 5.8).

— RegistryID SBFormat { get; set; } (see 5.13).

— byte[] BDBData { get; set; }.

— byte[] SBData { get; set; }.

5.4.3 Method summary

5.4.3.1 virtual BIR (byte[] record)

Description:	Constructs the BIR data from a byte array coded as a self-identifying record, as indicated in the relevant clauses of ISO/IEC 19785-3 and ISO/IEC 19785-4.
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 10.1).

5.4.3.2 virtual BIR (RegistryID bDBFormat, bool bDBEncription, bool bIRIntegrity, BiometricType bDBBiometricType, BiometricSubtype bDBBiometricSubtype, RegistryID bDBCaptureDevice, RegistryID bDBFeatureExtractionAlg, RegistryID bDBComparisonAlg, RegistryID bDBCompresionAlg, RegistryID bDBPADTechnique, byte[] bDBChallengeResponse, Date bDBCreationDate, byte[] bDBIndex, ProcessedLevel bDBProcessedLevel, RegistryID bDBProduct, Purpose bDBPurpose, byte bDBQuality, RegistryID bDBQualityAlg, List<Date> bDBValidityPeriod, Date bIRCreationDate, byte[] bIRCreator, byte[] bIRIndex, byte[] bIRPayload, byte[] bIRPointer, List<Date> bIRValidityPeriod, RegistryID sBFormat, byte[] bDBData, byte[] sBData)

Description:	Constructs the BIR data from its individual components.
Parameters:	— Each of the properties in the BIR class.
Exception:	If the input parameters are invalid, the format is not supported or operation fails due to error, BioAPIException (see 10.1).

5.4.3.3 virtual public byte[] ToArray()

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 10.1).

5.4.3.4 virtual void Dispose ()

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

5.5 Class BSPSchema [Serializable()]

5.5.1 Description

Represents the record in the component registry that defines the properties of the BSP installed in the system. Is a serializable class.

5.5.2 Properties Summary

— UUID BSPUUID {get;}.

— String BSPDescription {get;}: A NULL-terminated string containing a text description of the BSP.

— String Path {get;}: A pointer to a NULL-terminated string containing the path of the file containing the BSP executable code, including the filename. The path may be a URL. This string shall consist of ISO/IEC 10646 characters encoded in UTF-8 (see ISO/IEC 10646:2017, Annex D). 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.

— String SpecVersion {get;}: Major/minor version number of the BioAPI specification to which the BSP was implemented.

— String ProductVersion {get;}: The version string of the BSP software.

— String Vendor {get;}: A NULL-terminated string containing the name of the BSP vendor.

— List<RegistryID> BSPSupportedFormats {get;}: A list the data formats that are supported by the BSP (see 5.13).

— List<BiometricType> FactorsMask {get;}: A list of the biometric types supported by the BSP (see 5.7.2.2).

— List<BSPSchemaOperations> Operations {get;}: A list of the biometric operations supported by the BSP (see 5.7.2.4).

— List<BSPSchemaOptions> Options {get;}: A list of the biometric options supported by the BSP (see 5.7.2.5).

— int AdditionalDataPolicy {get;}: Threshold setting (maximum FMR value) used to determine when to release additionalData after successful verification.

— int MaxAdditionalDataSize {get;}: Maximum additionalData size (in bytes) that the BSP can accept.

— int DefaultVerifyTimeout {get;}: Default timeout value in milliseconds used by the BSP for Verify operations when no timeout is specified by the application.

— int DefaultIdentifyTimeout {get;}: Default timeout value in milliseconds used by the BSP for Identify and BioAPI_IdentifyMatch operations when no timeout is specified by the application.

— int DefaultCaptureTimeout {get;}: Default timeout value in milliseconds used by the BSP for Capture operations when no timeout is specified by the application.

— int DefaultEnrolTimeout {get;}: Default timeout value in milliseconds used by the BSP for Enrol operations when no timeout is specified by the application.

— int DefaultCalibrateTimeout {get;}: Default timeout value in milliseconds used by the BSP for sensor calibration operations when no timeout is specified by the application.

— int MaxBSPDbSize {get;}: 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.

— int MaxIdentify {get;}: Largest population supported by the identify function. Unlimited = 0xFFFFFFFF.

— int MaxNumEnrolInstances {get;}: The maximum number of distinct instances for which a BSP can create reference templates in one enrol operation. This information can be useful to an application that uses the application-controlled GUI feature.

— byte[] HostingEndpointIRI {get;}: An IRI identifying the framework for which the component registry contains a registration of the BSP. This parameter shall be ignored by frameworks conforming to this part of this International Standard 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.

— UUID BSPAccessUUID {get;}: 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 this International Standard and can be set to any UUID value by an application. It is provided to support interworking standards, which may specify the use of identical BSPs present on multiple computers from within an application running on the same or a different computer.

  NOTE 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).

— List<RegistryID> BSPSupportedAlgorithms {get;}: array of BioAPI_ALGORITHM_ID structures specifying the supported algorithms.

— List< UUID> BSPSupportedTransformOperations {get;}: array of BioAPI_UUID structures specifying the transform operations supported within the BioAPI_Tranform operation.

5.5.3 Method summary

5.5.3.1 virtual void Dispose ()

Description:	Removes all the information in the current object, leaving it empty for a next use.
Exception:	None.

5.6 Class Candidate

5.6.1 Description

Defines each of the resulting candidates from the Identify functionality.

5.6.2 Properties summary

— UUID Key { get; set; }: which defines the UUID of the candidate in the system (e.g. in the database).

— int FMRAchieved { get; set; }: which states the FMR achieved by the candidate during the Identify process.

5.7 Class DataTypes

5.7.1 Description

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

5.7.2 Enumerations

5.7.2.1 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:	— *NoValueAvailable* — *Left* — *Right* — *LeftThumb* — *LeftIndexFinger* — *LeftMiddleFinger* — *LeftRingFinger* — *LeftLittleFinger* — *RightThumb* — *RightIndexFinger* — *RightMiddleFinger* — *RightRingFinger* — *RightLittleFinger* — *LeftPalm* — *LeftBackOfHand* — *LeftWrist* — *RightPalm* — *RightBackOfHand* — *RightWrist*

5.7.2.2 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:	— *NoValueAvailable* — *MultipleBiometricTypes* — *Face* — *Voice* — *Finger* — *Iris* — *Retina* — *HandGeometry* — *SignatureOrSign* — *Keystroke* — *LipMovement* — *Gait* — *Vein* — *DNA* — *Ear* — *Foot* — *Scent*

5.7.2.3 BIRDatabaseAccess

Description:	Defines the access mode to the database.
Enum Constant Summary:	— *Read*: Access mode which allows only retrieval of records. — *ReadWrite*: 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.

5.7.2.4 BSPSchemaOperations

Description:	Enumerates the different operations that a BSP can offer to the biometric application (see 5.5).
Enum Constant Summary:	— *CalibrateSensor (0x00020000)* — *Capture (0x00000004)* — *CheckQuality (0x00080000)* — *ControlUnit (0x00400000)* — *CreateTemplate (0x00000008)* — *CreateTemplateWithAuxBIR (0x00000020)* — *EnableEvents (0x00000001)* — *Enrol (0x00000100)* — *GetIndicatorStatus (0x00010000)* — *Identify (0x00000080)* — *IdentifyAggregate (0x00000400)* — *PresetIdentifyPopulation (0x00001000)* — *Process (0x00000010)* — *ProcessWithAuxBIR (0x01000000)* — *QueryBFPs (0x00200000)* — *QueryUnits (0x00100000)* — *Security (0x10000000)* — *SetIndicatorStatus (0x00008000)* — *SetPowerMode (0x00004000)* — *Verify (0x00000040)* — *VerifyAggregated (0x00000200)* — *VerifyWithAuxBIR (0x02000000)*

5.7.2.5 BSPSchemaOptions

Description:	Enumerates the different options that can a handle a BSP (see 5.5).

Enum Constant Summary:	— *Adaptation (0x00000800)* — *AppGUI (0x00000010)* — *AchiveBFP (0x00020000)* — *Binning (0x00001000)* — *BirEncrypt (0x00000200)* — *BirSign (0x00000100)* — *CaptureMultiple (0x00400000)* — *CoarseScores (0x00100000)* — *ComparisonBFP (0x00040000)* — *GUIProgressEvents (0x00000020)* — *IdentifyIndicator (0x00200000)* — *OCC (0x00004000) (on-card comparison, formerly known as MOC)* — *AdditionalData (0x00000080)* — *ProcessingBFP (0x00080000)* — *ProcessMultiple (0x00800000)* — *QualityIntermediate (0x00000004)* — *QualityProcessed (0x00000008)* — *QualityRaw (0x00000002)* — *Raw (0x00000001)* — *SelfContainedDevice (0x00002000)* — *SensorBFP (0x00010000)* — *SourcePresent (0x00000040)* — *SubtypeToCapture (0x00008000)* — *TemplateUpdate (0x00000400)* — *Disabilities* — *PADFeature*

5.7.2.6 EventKind

Description:	Defines the kind of sources that can originate an event.
Enum Constant Summary:	— *Insert (0x00000001)* — *Remove (0x00000002)* — *Fault (0x00000004)* — *SourcePresent (0x00000008)* — *SourceRemoved (0x00000010)*

5.7.2.7 Facility

Description:	Defines originator of the error in an exception (see 10.1).
Enum Constant Summary:	— *Framework*: The error was reported by the framework component. — *BSP*: The error was reported by the biometric service provider. — *Unit*: The error was reported by the biometric unit.

5.7.2.8 GUIEnrolType

Description:	Indicates the enrol type of a BSP (see 10.2).
Enum Constant Summary:	— *TestVerify* — *MultipleCapture*

5.7.2.9 GUIMoment

Description:	Determines the moment when the processing of an operation is at the time of calling a GUI callback function (see 10.2).
Enum Constant Summary:	— *BeforeStart* — *AfterEnd*

5.7.2.10 GUIOperation

Description:	Determines the operation being performed when using GUI callback functions (see 10.2).
Enum Constant Summary:	— *Capture* — *Enrol* — *Identify* — *Verify*

5.7.2.11 GUIResponse

Description:	Enumeration of the possible actions to be performed by a BSP after a GUI event notification callback made by the BSP has returned control to the BSP (see 10.2).

Enum Constant Summary:	— *CycleStart* — *CycleRestart* — *Default* — *OpComplete* — *OpCancel* — *ProgressContinue* — *ProgressAbort* — *Recapture* — *SubOpStart* — *SubOpNext*

5.7.2.12 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 10.2).
Enum Constant Summary:	— *Capture* — *CreateTemplate* — *Identify* — *Process* — *Verify*

5.7.2.13 ProcessedLevel

Description:	Determines the level of processing of the BIR.
Enum Constant Summary:	— *Intermediate* — *Processed* — *Raw*

5.7.2.14 Purpose

Description:	Defines the purpose for which the BIR or process is intended.
Enum Constant Summary:	— *Verify* — *Identify* — *Enrol* — *EnrolForVerificationOnly* — *EnrolForIdentificacionOnly* — *Audit* — *Decide* — *NoPurposeAvailable*

5.7.2.15 ResultOptions

Description:	Defines the request to some BioAPI methods, to provide additional results to the originally defined (e.g. see 6.3.2).
Enum Constant Summary:	— *RequestAdaptedBIR*: Request that a BIR be constructed by adapting the reference template using the processed BIR that is the input to the biometric verification. — *RequestAdditionalData*: Request that the additionalData should be returned upon successful verification. — *RequestAdditionalData*: Request additional data to be used, for example, in an auditing process.

5.7.2.16 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. — *DigitalSignature (0x00000004)*: Indicates that the BioAPI Unit supports digital signature. — *ACBioGenerationWithMAC (0x00000010)*: Indicates that the BioAPI Unit supports ACBio generation using MAC. — *ACBioGenerationWithDigitalSignature (0x00000020)*: Indicates that the BioAPI Unit supports ACBio generation using digital signature.

5.7.2.17 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) — *QualityAssessment*: The unit is the collection of the different quality assessment processes. (0x00000010)

5.7.2.18 UnitIndicatorStatus

Description:	Defines possible values for the indicator status.
Enum Constant Summary:	— *Accept* — *Busy* — *Failure* — *Ready* — *Reject*

5.7.2.19 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.

5.8 Class Date

5.8.1 Description

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

5.8.2 Properties summary

— int DayOfMonth { get; set; }

— int Month { get; set; }

— int Year { get; set; }

— int Hour { get; set; }

— int Minute { get; set; }

— int Second { get; set; }

5.8.3 Methods summary

5.8.3.1 virtual bool IsLowerOrEqual (int day, int month, int year)

See 5.8.3.6.

5.8.3.2 virtual bool IsLowerOrEqual (int day, int month, int year, int hour, int minute, int second)

See 5.8.3.6.

5.8.3.3 virtual bool IsLowerOrEqual (Date date)

See 5.8.3.6.

5.8.3.4 virtual bool IsHigherOrEqual (int day, int month, int year)

See 5.8.3.6.

5.8.3.5 virtual bool IsHigherOrEqual (int day, int month, int year, int hour, int minute, int second)

See 5.8.3.6.

5.8.3.6 virtual bool 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*: Second for 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 otherwise.
Exception:	If the input parameters are invalid, the format is not supported or operation fails due to error, BioAPIException (see 10.1).

5.9 Class FrameworkSchema

5.9.1 Description

Defines the properties of the BioAPI Framework.

5.9.2 Properties summary

— UUID FrameworkUUID {get;}: UUID of the Framework component.

— String FwDescription {get;}: A NULL-terminated string containing a text description of the Framework.

— String Path {get;}: A pointer to a NULL-terminated string containing the path of the file containing the Framework executable code, including the filename. The path may be a URL. This string shall consist of ISO/IEC 10646 characters encoded in UTF-8 (see ISO/IEC 10646:2017, Annex D).

NOTE When 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.

— String SpecVersion {get;}: Major/minor version number of the BioAPI specification to which the Framework was implemented.

— String ProductVersion {get;}: The version string of the Framework software.

— String Vendor {get;}: A NULL-terminated string containing the name of the Framework vendor.

— UUID FwPropertyID {get;}: UUID of the format of the following Framework property.

— byte[] FwProperty {get;}: Address and length of a memory buffer containing the Framework property. The format and content of the Framework property can either be specified by a vendor or can be specified in a related standard.

5.9.3 Method summary

5.9.3.1 virtual void Dispose ()

Description:	Removes all the information in the current object, leaving it empty for a next use.
Exception:	None.

5.10 Class GUIBitmap

5.10.1 Description

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

5.10.2 Properties

— BiometricSubtype Subtype {get; set;}

— int Width {get; set;}

— int Height {get; set;}

— byte[][] Pixel {get; set;}

5.10.3 Method summary

5.10.3.1 virtual void Dispose ()

Description:	Removes all the information in the current object, leaving it empty for a next use.
Exception:	None.

5.11 Class Identifypopulation

5.11.1 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.

5.11.2 Properties summary

As mentioned, this interface has the following property, modified internally by the interface methods:

— List<PopulationMember> PopulationIdentified { get; }: This property can be used by the BSP to provide the list of members of the population.

5.11.3 Method summary

5.11.3.1 virtual 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 10.1).

5.11.3.2 virtual void Dispose ()

Description:	Removes all the information of the list of the population used for identification.
Exception:	BioAPIException (see 10.1).

5.11.3.3 virtual 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 otherwise.

5.11.3.4 virtual void Unbind ()

Description:	Ensures that all member BIRs are unbound.
Exception:	If operation fails, BioAPIException (see 10.1).

5.12 Class PopulationMember

5.12.1 Description

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

5.12.2 Properties summary

— UUID key { get; set; }: The user identifier, which is expected to be related (or the same) as the unique identifier of the user in the database.

— BIR Template { get; set; }: The user's biometric reference.

5.13 Class RegistryID

5.13.1 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 19785-3 and/or ISO/IEC 19785-4 BIRs.

5.13.2 Properties summary

— short Owner {get; set;}

— short Type {get; set;}

5.14 Class SecurityProfileType

5.14.1 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/validate the MAC or digital signature of the BIR and also to give 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.

5.14.2 Properties summary

— List<SecurityOptionsType> SupportedSecurityOptions {get;}

— byte[] ENCInfo {get;}

— byte[] MACInfo {get;}

— byte[] SIGNAlg {get;}: Identifies the digital signature algorithm supported by a BioAPI Unit. This BioAPI type shall be the XML value notation of ASN.1 identifier (see ISO/IEC 8824-1) assigned to digital signature algorithms.

— List<BSPSchemaOptions> ACBioOption {get;}: A mask which indicates which security options of MAC or digital signature are supported or to be executed by the BioAPI_Unit.

— byte[] HASHAlgForACBio {get;}.

— byte[] MACInfoForACBio {get;}

— byte[] SIGNAlgForACBio {get;}: As SIGNAlg but used to generate ACBio instances

5.14.3 Method summary

5.14.3.1 virtual void Dispose ()

Description:	Removes all the information in the current object, leaving it empty for a next use
Exception:	None.

5.15 Class UnitList

5.15.1 Description

Identifies a list of the selected BioAPI Units by category and ID. An object of this class shall only have a maximum of one UnitListElement per category. If no UnitListElement is found for a specific category, the BSP shall react properly (e.g. throwing an exception for not having a required Unit available).

5.15.2 Properties summary

— List<UnitListElement> { get; set; }: The list of the selected units.

5.15.3 Methods summary

5.15.3.1 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*: Category type and unitID of the new unit.
Exception:	If operation fails, BioAPIException (see 10.1).

5.15.3.2 int GetUnitID (UnitCategoryType unitCategoryType)

Description:	Gets the unitID of the unit which has the selected category type.
Parameters:	— *unitCategoryType*: Category of the unit for which the UnitID is asked.
Return Value:	Unit ID corresponding to the category type provided.
Exception:	If operation fails, BioAPIException (see 10.1). If there is no unit of the selected category type, BioAPIException of type BioAPIErrUnitCategoryNotFound.

5.15.3.3 virtual void Dispose ()

Description:	Removes all the information in the current object, leaving it empty for a next use.
Exception:	None.

5.16 Class UnitListElement

5.16.1 Description

Identifies a BioAPI Unit by category and ID. A list of these elements is used to establish the units to consider during the execution of aggregated functions.

5.16.2 Properties summary

— UnitCategoryType UnitCategory { get; set; }: The category of the unit.

— int UnitID { get; set; }: The ID assigned to the BioAPI_Unit.

5.17 Class UnitSchema

5.17.1 Description

Defines the properties of the biometric unit. There are no methods to modify the existing UnitSchema object.

5.17.2 Properties summary

— UUID BSPUUID { get; set; }: UUID of the BSP supporting this BioAPI Unit.

— UUID UnitManagerUUID { get; }: UUID of the software component directly managing the BioAPI Unit (may be either the BSP itself or a BFP).

— int UnitID { get; set; }: ID of the BioAPI Unit. This will be created by the BSP uniquely.

— UnitCategoryType UnitCategory { get; }: Defines the category of the BioAPI Unit.

— UUID UnitProperties { get; }: 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.

— String VendorInformation { get; }: Contains vendor proprietary information.

— List<EventKind> SupportedEvents { get; }: A mask indicating which types of events are supported by the hardware.

— UUID UnitPropertyID { get; }: UUID of the format of the following Unit property structure.

— byte[] UnitProperty { get; }: Address and length of a memory buffer containing the Unit property describing the BioAPI Unit. The format and content of the Unit property can either be specified by the vendor or be specified in a related standard.

— string HardwareVersion { get; }: A NULL-terminated string containing the version of the hardware. Empty if not available.

— string FirmwareVersion { get; }: A NULL-terminated string containing the version of the firmware. Empty if not available.

— string SoftwareVersion { get; }: A NULL-terminated string containing the version of the software. Empty if not available.

— string HardwareSerialNumber { get; }: A NULL-terminated string containing the vendor defined unique serial number of the hardware component. Empty if not available.

— bool AuthenticatedHardware { get; }: A boolean value that indicates whether the hardware component has been authenticated.

— int MaxBSPDbSize { get; }: Maximum size database supported by the BioAPI Unit, in case it is an Archive unit. If zero, no database exists.

— int MaxIdentify { get; }: Largest identify population supported by the BioAPI Unit, in case it is a Comparison unit. Unlimited = FFFFFFFF.

— List<SecurityProfileType> SecurityProfile { get; }: Security profile of the BioAPI Unit.

5.17.3 Method summary

5.17.3.1 virtual void Dispose ()

Description:	Removes all the information in the current object, leaving it empty for a next use.

Exception:	None.

5.18 Class UUID [Serializable()]

5.18.1 Description.

Throughout the use of this document, Unique Identifiers need to be used either for components or for users. This class defines the Universally Unique IDentifier (UUID) as the equivalent to the C# Guid data type. Is a serializable class.

5.18.2 Properties

— Guid ID { get; set; }

6 Object oriented interfaces for supporting BioAPI_Units

6.1 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, additionalData generated by the Verify method called previously.

Finally, in a sensor unit, the following fields should be implemented:

— byte[] AuxiliaryData, and

— UnitIndicatorStatus IndicatorStatus.

6.2 Interface IArchive

6.2.1 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), as far as the interface follows the specification of this document.

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.

6.2.2 Method summary

6.2.2.1 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 10.1).

6.2.2.2 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 10.1)

6.2.2.3 BIR GetSingleBIR (int unitID, UUID key)

Description:	Get the specified record.
Parameters:	— *unitID*: ID of the BioAPI unit to perform the operation. — *key*: UUID of the record to retrieve.
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 10.1).

6.2.2.4 List<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 10.1).

6.2.2.5 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 5.11 and 6.3).
Exception:	If the database has been closed, or the database was opened in write-only mode, or any other kind of error, BioAPIException (see 10.1).

6.2.2.6 IdentifyPopulation NewIdentifyPopulation (int unitID, List<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*: The list of UUIDs to be included into the new identify population.
Return Value:	The object representing the population (collection of BIRs) (see 5.11 and 6.3).
Exception:	If the database has been closed, or the database was opened in write-only mode, or any other kind of error, BioAPIException (see 10.1).

6.2.2.7 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 a specified criteria in the database.
Parameter:	— *unitID*: ID of the BioAPI unit to perform the operation. — *query*: The 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 5.11 and 6.3).
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 10.1).

6.2.2.8 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 10.1).

6.2.2.9 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*: The 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 mode, or any other kind of error, BioAPIException (see 10.1).

6.2.2.10 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 programme 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*: The specific BIR to store. — *UUID key*: The 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 10.1).

6.2.2.11 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 document). This information can be, for example, the demographics associated to the BiometricReference.
Parameters:	— *unitID*: ID of the BioAPI unit to perform the operation. — *biometricReference*: The 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 10.1).

6.2.2.12 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 programme 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 document). This information can be, for example, the demographics associated to the BiometricReference.
Parameters:	— *unitID*: ID of the BioAPI unit to perform the operation. — *biometricReference*: The specific BIR to store. — *auxiliaryData*: Additional data to be stored into the database in reference to the BIR stored. — *UUID key*: The 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 10.1).

6.3 Interface IComparison

6.3.1 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.

6.3.2 Method summary

6.3.2.1 BIR GetAdaptedBIR (int unitID)

Description:	If indicated by the VerifyResultOptions, this method returns 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:	The adapted BIR. If no adapted BIR is calculated, it returns NULL.
Exception:	If case of any error, BioAPIException (see 10.1).

6.3.2.2 int GetFMRAchieved (int unitID)

Description:	This function gets the FMR achieved during the previous comparison.
Parameters:	— *unitID*: ID of the BioAPI unit to perform the operation.
Return Value:	The FMR achieved.
Exception:	If case of any error, BioAPIException (see 10.1).

6.3.2.3 List<ICandidate> Identify (int unitID, int maxFMRrequested, BIR processedBIR, bool binning, int maxResults, int timeout)

See 6.3.2.4.

6.3.2.4 List<ICandidate> Identify (int unitID, int maxFMRrequested, BIR processedBIR, List<BIR> auxiliaryBIRs, bool binning, int maxResults, int timeout)

Description:	Performs biometric identification of the existing biometric sample. This function performs an identification (1-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 (see 5.4). — *auxiliaryBIRs*: (optional) the list of auxiliary BIRs used to improve the performance of the comparison algorithm (see 5.4). — *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 may improve the speed of the compare operation, it may also increase the probability of missing a candidate (due to the possibility of mis-binning and as a result, searching a bin which should, but does not, contain the matching BIR). — *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 10.1).

6.3.2.5 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 10.1).

6.3.2.6 bool Verify (int unitID, int maxFMRrequested, BIR processedBIR, BIR referenceTemplate, List<ResultOptions> options)

See 6.3.2.6.

6.3.2.7 bool Verify (int unitID , int maxFMRrequested, BIR processedBIR, BIR referenceTemplate, List<BIR> auxiliaryBIRs, List<ResultOptions> options)

Description:	Performs biometric verification of an existing biometric sample. This function performs a verification (1-to-1) 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 may not occur in all cases. If additionalData is associated with the reference template, 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 (see 5.4). — *auxiliaryBIRs*: (optional) the list of auxiliary BIRs used to improve the performance of the comparison algorithm (see 5.4). — *referenceTemplate*: the BIR to be verified against (see 5.4). — *Options*: requests additional output such as adapted BIR and/or 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 10.1).

6.4 Interface IProcessing

6.4.1 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.

6.4.2 Method summary

6.4.2.1 BIR CreateTemplate (int unitID, BIR capturedBIR, BIR referenceTemplate, RegistryID outputFormat, byte[] additionalData)

See 6.4.2.2.

6.4.2.2 BIR CreateTemplate (int unitID, List<BIR> capturedBIRs, BIR referenceTemplate, RegistryID outputFormat, byte[] additionalData, int unitID)

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 (see 5.4) — capturedBIRs: The list of captured BIRs (see 5.4) — referenceTemplate: optionally, the existing template to be updated (see 5.4) — 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 (see 5.13) — additionalData: additionalData that will be stored by the BSP.
Return Value:	Returns the BIR object that represents the result of processing (see 5.4)
Exception:	If the input parameters are invalid, the format is not supported or operation fails due to error, BioAPIException (see 10.1).

6.4.2.3 BIR Process (int unitID , BIR capturedBIR, RegistryID outputFormat)

See 6.4.2.4.

6.4.2.4 BIR Process (int unitID, BIR capturedBIR, List<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 (see 5.4). — *auxiliaryBIR*: a BIR containing the auxiliary data used in the operation (see 5.4). — *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 (see 5.13).
Return Value:	The BIR object that represents the result of processing (see 5.4).
Exception:	If the input parameters are invalid or operation fails due to error. BioAPIException (see 10.1)

6.4.2.5 byte AnalyseQuality (int unitID , BIR capturedBIR)

Description:	Processes the biometric sample captured to analyse its quality and return a quality score.
Parameters:	— *unitID*: ID of the BioAPI unit to perform the operation. — *capturedBIR*: the captured BIR (see 5.4).
Return Value:	The quality score, which could be any integer between 0 (lowest quality) and 100 (highest quality), and 255 if the quality algorithm failed to provide a score.
Exception:	If the input parameters are invalid or operation fails due to error, BioAPIException (see 9.1).

6.5 Interface ISensor

6.5.1 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.

6.5.2 Method summary

6.5.2.1 void Calibrate (int unitID , int timeout)

Description:	Performs a calibration of the referenced 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 10.1).

6.5.2.2 BIR Capture (int unitID , List<Purpose> purpose, BiometricSubtype subtype, RegistryID outputFormat, int timeout, List<ResultOptions> options)

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. — *subtype*: indicates the subtype of the biometric sample acquired. Null value indicates value is not provided (see 5.7.2.1). — *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 (see 5.13). — *timeout*: integer value specifying the timeout value in milliseconds. — *options*: requests additional output such as audit data.
Return Value:	The BIR object that represents the result of the capture operation (see 5.4).
Exception:	If the sensor device is busy or operation failed, BioAPIException (see 10.1).

6.5.2.3 UnitIndicatorStatus GetIndicatorStatus (int unitID)

Description:	This function gets the current status of the indicator.
Parameters:	— *unitID*: ID of the BioAPI unit to perform the operation.
Return Value:	The current status of the indicator for the sensor.
Exception:	If case of any error, BioAPIException (see 10.1).

6.5.2.4 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 10.1).

7 BFP level

7.1 Interface IBFP

7.1.1 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. Within the same BFP only one category of BioAPI_Units shall be included. It also may provide support to the communication with BSPs and the component registry.

7.1.2 Imported interfaces

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

— IArchive;

— ISensor;

— IProcessing;

— IComparison.

7.1.3 Properties summary

— IBFPSchema BFPSchema {get; set;}

— byte[] ACBioInstance { get; }

7.1.4 Events summary

There is a public event which allows the BFP to launch an event when something of interest occurs :

— event BFPEventCallback BFPEvent;

— event BFPGUIProgressCallback BFPGUICallback.

7.1.5 Method summary

In addition to the methods for the categoriy selected for the BSP (given in 6.2, 6.3, 6.4 or 6.5), the following methods shall be included in IBFP.

7.1.5.1 void BFPLoad (BFPEventCallback bfpNotifyCallback)

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 BSP's 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. 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:	— *bfpNotifyCallback*: defines a callback used to notify the BioAPI Framework of events.
Exception:	Thrown if any of the parameters is not valid or any other error during initialization. BioAPIException (see 10.1).

7.1.5.2 void BFPUnload ()

Description:	Disables events and de-registers the use of the current BFP in the BSP that has created a link to the BFP.
Exception:	Thrown if any of the parameters is not valid or any other error during initialization. BioAPIException (see 10.1).

7.1.5.3 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. NOTE This method is expected to forward the query to those relevant internal methods from the referenced BioAPI Unit. As the interoperability is only required at the BSP level, the way those relevant internal methods are called is implementation dependent, and therefore out of the scope of this document.

Parameters:	— *unitID*: ID of the BioAPI unit to perform the operation. — *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 10.1).

7.1.5.4 byte[] GetAuxiliaryData (int unitID)

Description:	Gets any auxiliary data available from the referenced BioAPI Unit (e.g. a Sensor), after performing an operation. If no auxiliary data is available, then the method shall return a NULL.
Parameters:	— *unitID*: ID of the BioAPI unit to perform the operation.
Return Value:	Data buffer containing the auxiliary data received from the BioAPI_Unit after processing the previous function. If no memory block has been allocated by the function, the value shall be NULL.
Exception:	If case of any error, BioAPIException (see 10.1).

7.1.5.5 List<UnitSchema> QueryUnits ()

See 7.1.5.6.

7.1.5.6 List<UnitSchema> QueryUnits (List<UnitCategoryType> unitCategories)

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 if the BFP is not reachable. BioAPIException (see 10.1).

7.1.5.7 void SetPowerMode (int unitID, UnitPowerMode powerMode)

Description:	This function sets the referenced BioAPI Unit of the loaded BSP to the requested power mode if the BioAPI Unit supports it.

Parameters:	— *unitID*: ID of the BioAPI unit to perform the operation. — *powerMode*: The indicator of the power mode to which the BioAPI_Unit is to be set.
Exception:	If case of any error, BioAPIException (see 10.1).

8 BSP level

8.1 Interface IBSP

8.1.1 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 also may 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, can 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 IBSPUnitSet interface.

8.1.2 Imported interfaces

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

— IArchive;

— ISensor;

— IProcessing;

— IComparison.

8.1.3 Properties summary

There is a public read-only property for stating the characteristics of the BSP:

— IBSPSchema BSPSchema {get; set;}

— byte[] ACBioInstance { get; }

8.1.4 Events summary

There is a public event which allows the BSP to launch an event when something of interest occurs:

— event BSPEventCallback BSPEvent.

8.1.5 Method summary

In addition to the methods for all the categories of BioAPI_Units that have been given in 6.2, 6.3, 6.4 and 6.5, the following methods shall be included in IBSP, although the developer may decide that a certain

number of them are not supported, returning when they are called the corresponding BioAPIException (see 10.1).

8.1.5.1 void BSPLoad (BSPEventCallback bspNotifyCallback)

Description:	Initializes the BSP. It shall not be called more than once without the corresponding call to BSPUnload()
Parameters:	— *bfpNotifyCallback*: defines a callback used to notify the BioAPI Framework of events.
Exception:	Thrown if any of the parameters is not valid or any other error during initialization. BioAPIException (see 10.1).

8.1.5.2 void BSPUnload ()

Description:	Disables events and de-registers the use of the current BSP.
Exception:	Thrown if any of the parameters is not valid or any other error during initialization. BioAPIException (see 10.1).

8.1.5.3 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*: The 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 10.1).

8.1.5.4 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. NOTE This method is expected to forward the query to those relevant internal methods from the referenced BioAPI Unit. As the interoperability is only required at the BSP level, the way those relevant internal methods are called is implementation dependent, and therefore out of the scope of this document.
Parameters:	— *unitID*: ID of the BioAPI unit to perform the operation. — *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 10.1).

8.1.5.5 UUID Enrol (UnitList unitList, BIR capturedBIR, BIR referenceTemplate, List<Purpose> purpose, BiometricSubtype subtype, RegistryID outputFormat, byte[] additionalData, List<ResultOptions> options)

See 8.1.5.10.

8.1.5.6 UUID Enrol (UnitList unitList, List<BIR> capturedBIRs, BIR referenceTemplate, List<Purpose> purpose, BiometricSubtype subtype, RegistryID outputFormat, byte[] additionalData, List<ResultOptions> options)

See 8.1.5.10.

8.1.5.7 UUID Enrol (UnitList unitList, int numberOfPresentations, int numberOfAttempts, BIR referenceTemplate, List<Purpose> purpose, BiometricSubtype subtype, RegistryID outputFormat, byte[] additionalData, int timeout, List<ResultOptions> options)

See 8.1.5.10.

8.1.5.8 UUID Enrol (UnitList unitList, BIR capturedBIR, UUID referenceID, List<Purpose> purpose, BiometricSubtype subtype, RegistryID outputFormat, byte[] additionalData, List<ResultOptions> options)

See 8.1.5.10.

8.1.5.9 UUID Enrol (UnitList unitList, List<BIR> capturedBIRs, UUID referenceID, List<Purpose> purpose, BiometricSubtype subtype, RegistryID outputFormat, byte[] additionalData, List<ResultOptions> options)

See 8.1.5.10.

8.1.5.10 UUID Enrol (UnitList unitList, int numberOfPresentations, int numberOfAttempts, UUID referenceID, List<Purpose> purpose, BiometricSubtype subtype, RegistryID outputFormat, byte[] additionalData, int timeout, List<ResultOptions> options)

Description:	Enrol a user by means of (in order of appearance in the list above): — Providing the sample to use for the enrolment in the first parameter of the method. — Providing a list of samples to use for the enrolment in the first parameter of the method. — 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 a previous enrolment. In the first case, the parameter ReferenceTemplate or ReferenceID shall have a Null value. In the second case, the biometric reference to be updated can be provided by means of a BIR (using ReferenceTemplate) or by means of the UUID (using ReferenceID).

	— The result of a successful enrolment will be the UUID assigned to the biometric reference created and, optionally, the BIR for that biometric reference, which will be located in the BiometricReference property. If for any reason (e.g. security concenrs) the BSP does not want to publish the generated biometric reference, that property shall be set to NULL. 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:	— *unitList*: BioAPI units to perform the operation. — *capturedBIR*: Optionally, the biometric sample to enrol, in BIR format (see 5.4). — *capturedBIRs*: Optionally, a list of BIRs with biometric samples to be used for the enrolment (see 5.4). — *numberOfPresentations*: When the sample to enrol is to be captured within the BSP, this parameter determines the number of samples to be acquired from the user. — *numberOfAttempts*: This parameter states the maximum number of attempts that has to be taken to acquire a Presentation, before returning an Exception related to Failure To Enrol (FTE). — *referenceTemplate*: Optionally, the BIR of the biometric reference to be updated (see 5.4). — *referenceID*: Optionally, the UUID of the biometric reference to be updated. — *purpose*: A value indicating the desired purpose (Enrol, EnrolForVerificationOnly, or EnrolForIdentification only). — *subtype*: Specifies which subtype (e.g., left/right eye) to enrol. A 'null' value indicates that the BSP is to select the subtype(s) (see 5.7.2.1). — *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 (see 5.13). — *additionalData*: A additionalData that will be stored by the BSP. — *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. — *options*: 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 10.1).

8.1.5.11 byte[] GetAuxiliaryData (int unitID)

Description:	Gets any auxiliary data available from the referenced BioAPI Unit (e.g. a Sensor), after performing an operation. If no auxiliary data is available, then the method shall return a NULL.
Parameters:	— *unitID*: ID of the BioAPI unit to perform the operation.
Return Value:	Data buffer containing the auxiliary data received from the BioAPI_Unit after processing the previous function. If no memory block has been allocated by the function, the value shall be NULL.
Exception:	If case of any error, BioAPIException (see 10.1).

8.1.5.12 List<ICandidate> IdentifyAggregated (UnitList unitList, int maxFMRrequested, BiometricSubtype subtype, bool binning, int maxResults, int timeout, List<ResultOptions> options)

See 8.1.5.12.

8.1.5.13 List<ICandidate> IdentifyAggregated (UnitList unitList, BIR inputBIR, int maxFMRrequested, BiometricSubtype subtype, bool binning, int maxResults, int timeout, List<ResultOptions> options)

Description:	This method provides an aggregated functionality. It increases the functionality of the Identify method from IComparison (see 6.3), 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 IComparison (6.3) 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*: BioAPI units to perform the operation. — *inputBIR*: The BIR to be identified, which could be in raw or processed format. — *subtype*: specifies which subtype (e.g., left/right eye) to capture. Null indicates that the value is not provided (see 5.7.2.1). — *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 Identify methods in IComparison (6.3).
Return Value:	See the Identify methods in IComparison (6.3).
Exception:	In case of any of the Capture exceptions (6.5) or the Identify exceptions (6.3), BioAPIException (see 10.1).

8.1.5.14 List<BFPListElement> QueryBFPs ()

See 8.1.5.15.

8.1.5.15 List<BFPListElement> QueryBFPs (List<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 has been unloaded, or any other error. BioAPIException (see 10.1).

8.1.5.16 List<UnitSchema> QueryUnits ()

See 8.1.5.17.

8.1.5.17 List<UnitSchema> QueryUnits (List<UnitCategoryType> unitCategories)

Description:	Returns Units available by the BSP. All Units in a BSP 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 being used.
Exception:	Thrown if the method is called after the BSP has been unloaded. BioAPIException (see 10.1).

8.1.5.18 bool VerifyAggregated (UnitList unitList, int maxFMRrequested, BIR referenceTemplate, BiometricSubtype subtype, int timeout, List<ResultOptions> options)

See 8.1.5.21.

8.1.5.19 bool VerifyAggregated (UnitList unitList, int maxFMRrequested, BIR inputBIR, BIR referenceTemplate, BiometricSubtype subtype, int timeout, List<ResultOptions> options)

See 8.1.5.21.

8.1.5.20 bool VerifyAggregated (UnitList unitList, int maxFMRrequested, UUID referenceKey, BiometricSubtype subtype, int timeout, List<ResultOptions> options)

See 8.1.5.21.

8.1.5.21 bool VerifyAggregated (UnitList unitList, int maxFMRrequested, BIR inputBIR, UUID referenceKey, BiometricSubtype subtype, int timeout, List<ResultOptions> options)

Description:	This method provides an aggregated functionality. It increases the functionlality of the Verify method from IComparison (see 6.3), allowing the verify operation in the following cases:

	— The biometric sample is captured directly by the BSP calling its own Sensor unit. Therefore, the inputBIR parameter is not present. — The biometric sample is supplied either in raw or processed format. — The biometric reference is declared by its UUID. Therfore the referenceKey parameter is added. — The biometric reference is provided as a BIR. Therefore, the referenceTemplate parameter is used. — The 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 IComparison (6.3) 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*: BioAPI units to perform the operation. — *inputBIR*: the sample to be verified, either in raw or processed format. — *referenceKey*: the UUID of the biometric reference to be used for verification. — *referenceTemplate*: the BIR corresponding to the biometric reference to be used for verification. — *subtype*: specifies which subtype (e.g., left/right eye) to capture. Null indicates that the value is not provided (see 5.7.2.1). — *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 IComparison (6.3).
Return Value:	See the Verify methods in IComparison (6.3).
Exception:	In case of any of the Capture exceptions (6.5) or the Verify exceptions (6.3), BioAPIException (see 10.1).

8.1.5.22 void SetPowerMode (int unitID, UnitPowerMode powerMode)

Description:	This function sets the referenced BioAPI Unit of the loaded BSP to the requested power mode if the BioAPI Unit supports it.
Parameters:	— *unitID*: The ID of the BioAPI Unit selected. — *powerMode*: The indicator of the power mode to which the BioAPI_Unit is to be set.
Exception:	If case of any error, BioAPIException (see 10.1).

8.1.5.23 void SubscribeToGUIEvents (GUISelectEventCallback guiSelectEventCallback, GUIStateEventCallback guiStateEventCallback, GUIProgressEventCallback guiProgressEventCallback)

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:	— *guiSelectEventCallbackFunction*: specifies address to the callback function for the Select Event. — *guiStateEventCallbackFunction*: specifies address to the callback function for the State Event. — *guiProgressEventCallbackFunction*: specifies address to the callback function for the Progress Event.
Exception:	BioAPIException (see 10.1).

8.1.5.24 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 10.1).

9 Framework level

9.1 Interface IComponentRegistry

9.1.1 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 at the "Component registry interface" clause in ISO/IEC 19784-1.

9.1.2 Method summary

9.1.2.1 void InstallBFP (BFPSchema bfpSchema, bool 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 10.1).

9.1.2.2 void InstallBSP (BSPSchema bspSchema, bool 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 10.1).

9.1.2.3 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 10.1).

9.1.2.4 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 10.1).

9.2 Interface IFramework

9.2.1 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.

9.2.2 Inherited interfaces

The IComponentRegistry 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.

9.2.3 Properties summary

There is a public read-only property for stating the characteristics of the BSP:

— IFrameworkSchema FrameworkSchema { get; }.

9.2.4 Method summary

9.2.4.1 void BSPLoad (UUID bspID, BFPEventCallback notifyCallback, BFPEnumerationCallback bfpEnumerationCallback, String context)

Description:	Initializes a BSP. Initialization includes enabling all events except those that were disabled by the most recent call to EnableEventNotifications. The application can choose to provide an event handler function to receive notification of events. Many applications can independently and concurrently load the same BSP, and each application can establish its own event handler. They will all receive notification of an event. The same or different event handlers can be used if an application loads multiple BSPs. An application may establish as many event handlers as it wishes, for a given BSP, by calling BioAPI_BSPLoad one or more times for that BSP. An event handler is identified by a combination of address and context. When an event occurs in a BSP, the BSP may send an event notification to the Framework by calling the Framework's event handler. When the Framework receives an event notification from a BSP, it shall send one notification to each event handler established by each application for which that event notification is enabled for that BSP. Therefore, a single event notification callback made from a BSP to the Framework may result in zero or more callbacks made by the Framework to zero or more applications. When the framework receives an event notification from a BSP, it shall call all the event handlers established by each application for that BSP. If an application has set up multiple event handlers, they shall be called one at a time (in any order chosen by the Framework) rather than concurrently. Event notification may occur at any time, either during a BioAPI call (related or unrelated to the event) or while there is no BioAPI call in execution. Application writers should ensure that all callbacks are properly and safely handled by the application, no matter when the application receives them. NOTE This usually requires the use of thread synchronization techniques and discipline in the actions performed by the application code placed in event handlers.

	There is a "use count" on the establishment of event handlers; they have to be de-established (by BSPUnload) as many times as they were established. When the BioAPI Framework calls BSPLoad, it receives from the BSP an “insert” event notification for each available BioAPI unit (of each category). If the biometric application has provided an event handler in the call to BSPLoad and has not disabled “insert” event notifications, the Framework will call back, in turn, the application's event handler. Insert notifications can be disabled by calling the function EnableEventNotifications. This will indicate to the biometric application that it can go ahead. If the hardware component for a specific functionality is not present, the "insert" event cannot be raised until the hardware component has been plugged in. This function shall only be called if there is at least one call to Init for which a corresponding call to Terminate has not yet been made. The BSPLoad function is not to be called unless the BSP has been installed using InstallBSP. A determination of installed BSPs can be made through a call to EnumBSPs.
Parameters:	— *bspID*: Identifies the BSP to initialize. — *notifyCallback*: The event notification function provided by the caller. This defines the callback for event notifications from the loaded biometric service provider. — *bfpEnumerationCallback*: The event notification function provided by the caller. The callback function allows a BSP to know which BFPs are instalated in the biometric system. — *context*: Generic pointer to context information. When the selected biometric service provider raises an event, this value is passed as input to the event handler specified by NotifyCallback.
Exception:	Thrown if the framework has been terminated or any other error. BioAPIException (see 10.1).

9.2.4.2 void BSPUnload (UUID bspID, String context)

Description:	De-registers event notification callbacks for the caller identified by BSPUUID. BSPUnload is the analogue call to BSPLoad. If all callbacks registered with BioAPI are removed, then BioAPI unloads (for that biometric application) the BSP that was loaded by calls to BSPLoad. The BioAPI Framework uses the three input parameters to uniquely identify registered callbacks. This function shall only be called (for a given BSP UUID) if there is at least one call to BSPLoad (for that BSP UUID) for which a corresponding call to this function has not yet been made. This includes the case in which the actions relative to a missing call to BSPUnload are implicitly performed by the BioAPI Framework during a call to Terminate.
Parameters:	— *bspID*: Identifies the BSP to initialize. — *context*: Generic pointer to context information. When the selected biometric service provider raises an event, this value is passed as input to the event handler specified by NotifyCallback.
Exception:	Thrown if the framework has been terminated or any other error. BioAPIException (see 10.1).