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)

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.