INTERNATIONAL STANDARD

ISO/IEC 30106-1

First edition
2016-03-15

Information technology — Object oriented BioAPI —

Part 1: Architecture

Technologies de l'information — Objet orienté BioAPI —

Partie 1: Architecture

Reference number
ISO/IEC 30106-1:2016(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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

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

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

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, SC 37, *Biometrics*.

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

— *Part 1: Architecture*

— *Part 2: Java implementation*

— *Part 3: C# implementation*

Introduction

The existing versions of BioAPI (ANSI version-INCITS 358 and ISO/IEC 19784-1) specify an application programming interface expressed in the C language. The use of a portable language like C ensures the BioAPI is accessible across multiple computing platforms and application domains. BioAPI is an appropriate fit for applications written in C and is adequate for applications written in C++.

Unfortunately, a function-based language like C does not map easily to the object oriented domain where this issue may be answered with an object oriented (OO) version of BioAPI. As noted, the function-based nature of a C API does not map easily to the object oriented paradigm (i.e. languages such as C# and Java). In particular, the use of a C API from within an object oriented application is unnatural and requires programming constructs which introduce complexity to the development of an application. Development of a OO version of BioAPI aims to increase the productivity of software practitioners who who wish to use the BioAPI whilst remaining in the object oriented domain.

A standard object oriented version of BioAPI allows, in case of Java, BSPs that are intended for loading into a Java-based application server to perform verification and/or identification operations. In those application servers, use of the OO BioAPI is more natural when developing a framework and BSPs than the C version of BioAPI.

Another area in which a standard OO version of BioAPI would be useful is that of small computing devices based on an object oriented language (OOL), where (as on the large application servers mentioned above) an OO BioAPI framework and OO BSPs would fit better than their C counterparts.

This part of ISO/IEC 30106 is expected to have the following impact:

— enable creation of BioAPI applications by developers more comfortable with Object Oriented Languages;

— create a market of standard OO BSP components which target OO environments such as Java application servers, Java applets, small Java devices, .NET servers, .NET applications, web services;

— increase the level of adoption of BioAPI by decreasing the barrier of entry for OO developers. This includes providing access to C based BSPs (as if they were OO BSPs) through special versions of the BioAPI framework, bridging a standard OO BioAPI framework to a standard C BioAPI framework.

Information technology — Object oriented BioAPI —

Part 1:
Architecture

1 Scope

This part of ISO/IEC 30106 specifies an architecture for a set of interfaces which define the OO BioAPI. Components defined in this part of ISO/IEC 30106 include a framework, Biometric Service Providers (BSPs), Biometric Function Providers (BFPs) and a component registry.

NOTE Each of these components have an equivalent component specified in ISO/IEC 19784-1 as the OO BioAPI is intended to be an OO interpretation of this part of ISO/IEC 30106.

For this reason, this part of ISO/IEC 30106 is conceptually equivalent to ISO/IEC 19784-1. Concepts present in this part of ISO/IEC 30106 (for example, BioAPI_Unit and component registry) have the same meaning as in ISO/IEC 19784-1. While the conceptual equivalence of this part of ISO/IEC 30106 will be maintained with ISO/IEC 19784-1, there are differences in the parameters passed between functions and the sequence of function calls. These differences exist to take advantage of the features provided by Object Oriented Programming Languages.

2 Normative references

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

ISO/IEC 2382-37, *Information technology — Vocabulary — Part 37: Biometrics*

ISO/IEC 19784-1:2006, *Information technology — Biometric Application Programming Interface — Part 1: BioAPI Specification*

ISO/IEC 19785-1:2015, *Information technology — Common Biometric Exchange Formats Framework — Part 1: Data Element Specification*

ISO/IEC 19785-3:2015, *Information technology — Common Biometric Exchange Formats Framework — Part 3: Patron format specifications*

3 Terms and definitions

For the purposes of this document, the terms and definitions in ISO 2382-37, ISO/IEC 19784-1, and ISO/IEC 19785 (all parts) apply.

4 Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms defined in ISO/IEC 19784-1, ISO/IEC 19785 (all parts) and the following apply.

API	Application Programming Interface
BDB	Biometric Data Block
BFP	Biometric (OO) Function Provider

BIR	Biometric Information Record
BSP	Biometric Service Provider
CBEFF	Common Biometric Exchange Formats Framework
FMR	False Match Rate
FPI	Function Provider Interface
GUI	Graphical User Interface
ID	Identity/Identification/Identifier
MOC	Match on Card
OOBioAPI	Object Oriented BioAPI
PID	Product ID
SB	Security Block NOTE This term and abbreviation is imported from ISO/IEC 19785-1.
SBH	Standard Biometric Header NOTE This term and abbreviation is imported from ISO/IEC 19785-1.
SPI	Service Provider Interface
UUID	Universally Unique Identifier

5 Object Oriented BioAPI architecture

5.1 Summary of BioAPI architecture

Object Oriented BioAPI shall be defined in a way that allows both structured development of applications and services, as well as interoperability between application and BSPs, and among BSPs. Therefore, the definition has to be done respecting the hierarchical structure of BioAPI (19784-1). In few words, an application shall be developed using an OO BioAPI that allows the instantiation of a BSP, which is based on the instantiation of one or several BioAPI_Units. The BSP can host more than one BioAPI_Unit of each category, and several units of each type can be used at any time, no presenting any kind of limitation on the units to be used in a BSP. This makes it necessary for all methods to provide the reference to the unit to be used in each of the operations.

An application is not permitted to directly access BioAPI_Units. Therefore, this part of ISO/IEC 30106 does not define a BioAPI_Unit interface, but only an object oriented hierarchical interface/class model that may ease the implementation of the BSP. The BSP shall inherit all public methods and data structures for each BioAPI_Unit the BSP encapsulates. It is the responsibility of the BSP's implementation to determine which functions of the BioAPI_Unit are offered to clients of the BSP. Note, an exception should be returned for any BSP interface method that is not implemented in the BSP. This is represented in Figure 1.

Programming an application, which interacts directly with a BSP, introduces practical considerations for the application designer. One such consideration is dealing with third-party suppliers who provide biometric components for use in applications. For example, a supplier of sensors may desire to include support for their entire family of sensors as a single entity (e.g. a library). To achieve this, the supplier may consider implementing the library as a single BSP, but may choose to not implement monolithic methods (i.e. aggregated functionality such as Enrol, which does in one single call the capture, the processing of the sample, the creation of the biometric reference and even the archiving). Without monolithic methods, the BSP forces the application to take responsibility for the implementation of monolithic style functionality. This situation means the BSP may pass biometric data back to the application so the application may perform processing. The application may then need to pass this data on to another BSP for additional processing. In some contexts, the passing of data out to an application may simply be inefficient, in other contexts this may be a security concern. A possible solution for this inconvenience is to allow the hypothetical sensor BSP to interact directly with other BSPs, rather than involving the application in the management of this communication. To achieve this, the biometric product provider may create an entity (e.g. a library) containing several BioAPI_Units of the same kind. This is called a Biometric Function Provider (BFP) and exhibits the following characteristics.

— The BFP shall only host BioAPI_Units of the same category.

— The BFP is meant to allow that a BSP is linked to one of its BioAPI_Units, in order to complete or adapt the functionality of the BSP.

— The BFP shall not provide functionality directly to the application, but only link to the BSP. The BSP communicates with the BFP on behalf of the application. The BSP is responsible for providing the BFP's functionality to the application.

The above mentioned situation also solves a problem from developers' point of view, which deals with simplicity in developing applications. If an application may require to use BioAPI_Units from different providers (e.g. a sensor from one provider and processing, comparison and archive BioAPI_Units from other provider), then it will have to load two different BSPs, calling each of the methods independently. As mentioned earlier, this has the inconvenience that it won't be possible to call a monolithic method, such as Verify(), which performs the data capture, the processing, extraction of the biometric reference from the database, the comparison and taking the decision, all within the same single call. Then, the application programmer will have to know which individual methods have to be called from each of the BSPs, in order to get the same functionality. To summarize, a level of abstraction and efficiency is achieved through the use of BFPs to segregate functionality into discrete sets. Each BFP aggregates a set of functionality provided by a product supplier (through a BioAPI_Unit). These BFPs are used by a BSP, where the BSP combines functionality from each BFP to offer monolithic methods for use by a client. The client application may then call the monolithic methods in the BSP without concern for the underlying implementation offered by a product supplier.

Another consideration for an application designer is the concern about security in regards to certain operations. Biometric data is typically sensitive personal data and some implementations of OO BioAPI may require the client application is not capable of directly accessing this data (ie: BIRs). By taking advantage of BFPs, and the abstraction facilities they offer, the possibility of tampering by malware or malicious users at the application level is removed. By using BFPs, all sensitive data will be handled at a level no higher than the BSP level. This means no BIR will be accessible to the application, not only simplifying application development, but eases concerns in relation to security.

The BFP shall not be accessed directly by the application. BioAPI calls are created to allow the application to know the BioAPI_Units that are contained in the BFPs so that the application may later request one BSP to include one of those BioAPI_Units of the BFP. This is done through a component registry and the procedure is further described in 5.4.

All the above ideas shall be implemented to allow the dynamic selection of components to be used by the application, but with no a-priori knowledge from the application developer. This is achieved by the inclusion of a common framework, which shall be installed in the system where the application is expected to be running. Then, the application shall request the framework for the list of the BSPs and BFPs installed, select the BSPs (with the requested selection of BioAPI_Units from BFPs) to be instantiated dynamically by the framework, and then accessing their methods and data structures through the framework. The application shall never be allowed to access the BSPs directly. This is summarized in Figure 1.

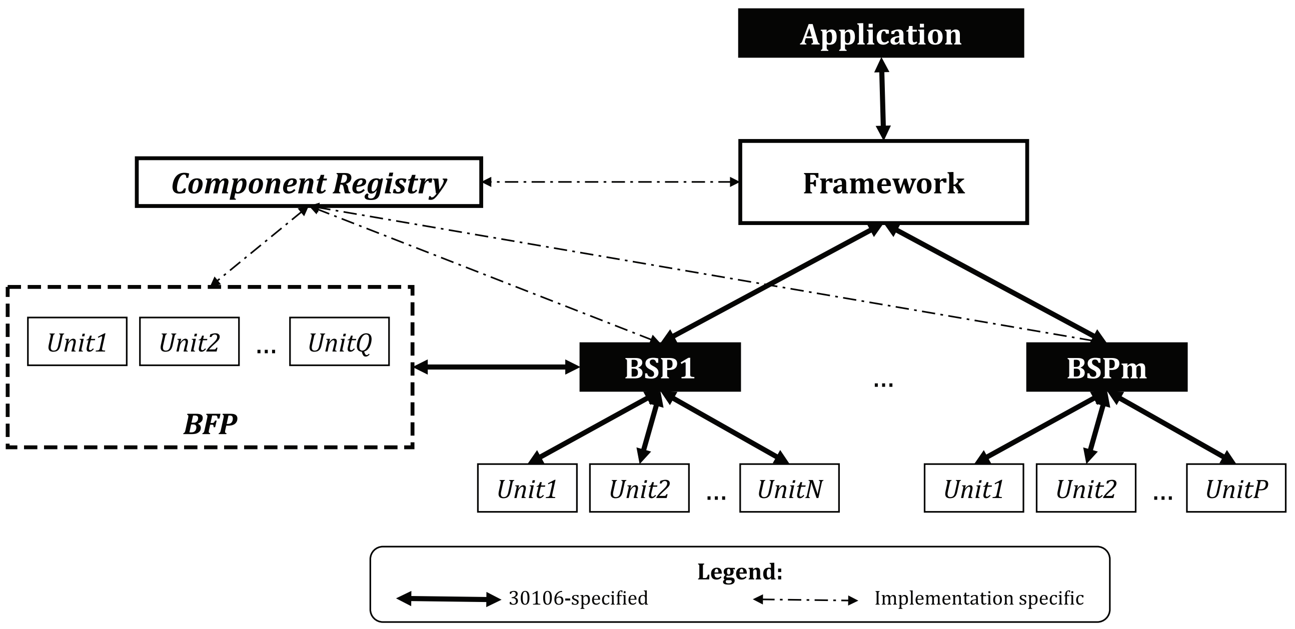

Figure 1 — Generic structure of a framework-based application

BSPs may be accessed by several applications at the same time, and it may also happen that BioAPI_Units in the BFPs are also accessed by several BSPs at the same time. It is up to the implementation of the framework the way that this implemented (e.g. this may be done by queuing requests from the different sources, or by responding to occurring events).

For those developers that are familiar with ISO/IEC 19784, it is important to provide a relationship about the interfaces there defined and the different layers that compose these International Standards. The correspondence between both International Standards is given in Table 1.

Table 1 — Correspondence between ISO/IEC 30106 and ISO/IEC 19784 interfaces

Interface in ISO/IEC 30106	Interface in ISO/IEC 19784
IFramework	API
IBSP	SPI
IBFP	SFPI

Along the ISO/IEC 30106 series, the Unit level will be also specified, not as for specifying a different interface, but for structuring in a hierarchical way the support for each of the categories of BioAPI_Units at the IBSP or IBFP interfaces.

5.2 BioAPI compatibility requirements

This part of ISO/IEC 30106 is defined as compatible with BioAPI version 3.0 (ISO/IEC 19784-1.revision 1), but the framework may be developed as to allow the cross compatibility with previous versions of BioAPI, as to allow BioAPI 2.x compliant BSPs and BFPs to be used in an ISO/IEC 30106 compliant application.

There is no provision to allow Object Oriented BSPs to be used in an ISO/IEC 19784-compliant framework and application.

5.3 Graphical User Interface (GUI)

A BSP may handle by itself, the interaction between the user and the system. For example, a BSP with a sensor BioAPI_Unit may address all the interaction by the displaying messages, illuminating LEDs and/or activating sounds at the sensor itself. Such kind of interaction may include providing feedback to the user on

— the finger to be placed at the sensor,

— how to improve the location of the sample in the sensor (e.g. alignment of the biometric characteristic in relation to the sensor active area),

— whether the acquisition has been done in a correct way, or

— whether the biometric comparison has led to a positive or negative decision.

But in certain cases, it may be requested that such interaction is handled by the application using such BSP, showing all messages and illustrations in the computer screen. This may also be handled by the BSP itself, but this can lead to certain kinds of inconveniences such as the following:

— the BSP graphical user interface may not be adaptable to the different sizes and resolutions that the application screen is using;

— the feedback provided has to follow the same look and feel as the main application, including the use of service provider logos;

— the feedback has to be adaptable to the language of the user being identified, or where the system is deployed.

In those cases, it is recommended in order to avoid re-programming the BSP or increasing the complexity of the BSP, the interaction is handled by the application itself. As the feedback depends on the specific step that the BSP is executing, and those steps are mostly controlled by the BSP (i.e. most BSP functions are offered to the application as monolithic methods), then the way to solve this is by using callback functions.

Callback functions shall be developed for each step where a BSP is required to either interact with, or provide feedback to, a user. Naturally, the application shall provide a set of functions the BSP's callbacks will map to, which the BSP will call within the context of the original method call.

Therefore, if the BSP allows the use of callback functions for handling the GUI, the following development actions shall be taken, for each of the processes requiring such interaction (e.g. Capture).

— The application shall implement all GUI related functions for such process. These functions are of the following kinds:

  — Select Function: to show the interaction with the user at the beginning or the end of the whole process.

  EXAMPLE At the beginning the process, the following message may be provided: "Capture process for the index finger of the right hand". And at the end of the process, the message to be given could be: "Acquisition finished. Thank you!". Therefore, the callback function shall know if it is being called at the beginning of the process, or at the end. Also, the callback function can also provide the BSP with information about a user action, such as cancelling the capture process. All this information flow is provided in parameters of the callback function.

  — State Function: Sometimes the BSP process may take several internal steps, which can be designed as different states. Therefore, the state function will provide feedback depending on whether a certain internal step is to be started or finished. This leads to the need for the State Function to know which internal step is being processed, and that information will be provided as parameters to the callback function. Also, feedback provided by the user will be passed to the BSP by the parameters involved, such as cancelling the whole process.

  — Progress Function: In some other cases, it may be necessary to provide the user with real time feedback on the progression of the process.

  EXAMPLE A system may be showing in the screen the live image of the fingerprint or face within the acquisition process. In such case, a Progress callback function may be implemented to show the data stream.

— Once the callback functions have been implemented for the process, the application shall report the BSP about the callback functions to be used, before calling the process. This will be done by subscribing the BSP to those selected GUI events, and the relevant callback functions.

— Once subscribed, the application can call the process (e.g. Capture).

— Finally, when the process has been completed, the application can unsubscribe the BSP from the previously subscribed events.

NOTE Further information can be found in ISO/IEC 19784-1.

5.4 Implementation guidelines

5.4.1 Basic concepts

A common scenario for BioAPI's implementation is one where a framework is developed. A BioAPI framework provides a further abstraction, on top of BSPs and BFPs, allowing an application developer to use biometric components (BSPs or BFPs) without concern for their underlying implementation. In a framework approach, it is the responsibility of the application developer to determine which functions offered by BSPs or BFPs are available to an end-user.

In an Object Oriented BioAPI implementation, three developer roles are involved in the use of this part of ISO/IEC 30106. The following points provide guidance on which clauses are applicable to each developer role:

— Developers providing services (ie: BSPs or BFPs) for access by biometric applications, will find 5.4.2, 5.4.3 and 5.4.4 are applicable

— Framework developers involved with integrating the framework into a system and making BSPs and BPFs accessible to such systems will find 5.4.5 applicable

— Application developers who directly use third party BSPs will find 5.4.6 is applicable to this group of developers.

5.4.2 BioAPI_Unit development

BSPs and BFPs will implement BioAPI_Units. They shall be developed including all properties and methods defined in this International Standard. If when implementing a certain BioAPI_Unit, the developer does not want to provide a certain functionality, then, the following indications shall be followed.

— For each of the methods, including the overloading of them, that are not to be supported, the method shall be programmed and published, but containing only the throw of a BioAPIException indicating that such function is not supported.

— Whatever public property is involved only in the non-supported functionality, its value shall be set to NULL.

— The UnitSchema shall provide an accurate information on the functionality supported by the BioAPI_Unit.

It is necessary to remark that the implementation of BioAPI_Units is entirely proprietary and is strictly supported and implemented by BioAPI_BSP developer.

5.4.3 BFP development

As it was mentioned in 5.1, a Biometric Function Provider (i.e. BFP), is a set of BioAPI_Units of the same category (i.e. either archive, comparison, processing or sensor), that are provided for the BSPs, so that