DRAFT INTERNATIONAL STANDARD

ISO/IEC DIS 30106-1

ISO/IEC JTC **1**/SC **37** Secretariat: **ANSI**

Voting begins on:
2014-12-04

Voting terminates on:
2015-03-04

Information technology — BioAPI for object oriented programming languages —

Part 1: Architecture

Titre manque

ICS: 35.040

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENT AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

Reference number
ISO/IEC DIS 30106-1:2014(E)

Copyright notice

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 30106-1 was prepared by Technical Committee ISO/TC JTC1, *Information Technology*, Subcommittee SC 37, *Biometrics*.

This second/third/... edition cancels and replaces the first/second/... edition (), [clause(s) / subclause(s) / table(s) / figure(s) / annex(es)] of which [has / have] been technically revised.

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, the ANSI version-INCITS 358 and ISO/IEC 19784-1, specify an application programming interface expressed in the C language. The use of this language ensures the wide applicability and implementability of BioAPI across multiple computing platforms and application domains. This API is an appropriate fit for applications written in the same language, and is adequate for applications written in C++.

Lack of portability of applications written in C is an issue that can be answered with an object oriented (OO) version of BioAPI. A C API does not work well with applications written in Java, C#, and other programming languages. In particular, the use of a C API from within an Object oriented (apart from C++) application is very unnatural, and requires certain programming artifices that introduce complications in the application, thus increasing the cost of application development and maintenance. If a standard OO version of BioAPI were available, the development of Java, C#, etc. applications that use a standard biometric API would be easier and cheaper than it is today.

A standard Object Oriented version of BioAPI would also allow, 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, a Java BioAPI framework and Java BSPs would work better than their C counterparts.

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

This standard is expected to have the following impact:

- Enable creation of many new BioAPI applications by developers more comfortable with Object Oriented Languages.

- Create a market of OO components that are standard OO BioAPI BSPs, and that can be used (along with a OO BioAPI framework) in OO environments such as Java-based application servers, Java applets, or small Java-based devices, .NET servers, .NET applications, Web Services;and

- Increase the level of adoption of BioAPI by enabling OO application developers to access C BSPs (as if they were OO BSPs) through special versions of the BioAPI framework that will bridge a standard OO BioAPI framework to a standard C BioAPI framework.

Information Technology — BioAPI for Object Oriented Programming Languages — Part 1: Architecture

1 Scope

The proposed standard will specify a generic architecture for the interface of a BioAPI OO framework and BioAPI OO BSP which will mirror the corresponding components specified in ISO/IEC 19784-1. Therefore, the position occupied by the proposed standard within the general picture of biometrics standards will be the same position that ISO/IEC 19784-1 occupies, the only difference being the programming language of the interfaces. The concepts such as BioAPI_Unit, component registry, etc. are present in this standard and will have the same meaning as in ISO/IEC 19784-1. The semantic equivalence of this standard will be maintained with ISO/IEC 19784-1, but there are differences in actual parameters passed between functions and the sequence of function calls. These differences exist to take advantage of the object oriented benefits of Object Oriented Programming Languages.

2 Conformance

Annex A specifies the conformance requirements for systems/components claiming conformance to this standard.

3 Normative references

The following referenced documents are indispensable for the application 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 19784-1:2006, Information Technology – Biometric Application Programming Interface – Part 1: BioAPI Specification

ISO/IEC 19785-1:2006, Information Technology – Common Biometric Exchange Formats Framework – Part 1: Data Element Specification

ISO/IEC 19785-2:2006, Information Technology – Common Biometric Exchange Formats Framework – Part 2: Procedures for the Operation of the Biometric Registration Authority

4 Terms and definitions

For the purposes of this document, the terms defined in ISO/IEC 19784-1, ISO/IEC 19785 and ISO 2382-37 apply.

5 Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms defined in ISO/IEC 19784-1, ISO/IEC 19785 and the following applies:

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

6 Object Oriented BioAPI architecture

6.1 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, this has to allow the development of applications in a system based on a BioAPI Framework, as well as applications in a system where no Framework is used (framework-free). Starting with the latter, and application in a framework-free based system 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 when a session is attached, several units of each type can be selected, 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.

It is not allowed that the application access BioAPI_Units directly. Therefore this International Standard 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 of the BioAPI_Units, and it is up to the implementation of the BSP implementation to decide which of them are offered to the external world, and which of them are only returning the lack of support of such method. This is represented in the following figure.

Programming an application using a direct connection with the BSPs, rises certain concerns in practical situations. The first concern is dealing with industry providing elements to be used in applications. It may happen that a provider is only providing sensors, and would like to include support to all its family of sensors as a single entity (e.g. a library). That provider may consider implementing that as a single BSP, but not being interested in providing 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). Therefore it will need to request these kind of functions to be done at the application level, exchanging biometric information between the BSP and the application. Then, the application may forward that biometric data to another BSP to complete the action. So a possible solution for this inconvenience is to allow those sensors to directly interact with other BSPs, instead of using the application for that. 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), which has mainly 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 to the application, but only the link to the BSP. It is the BSP the one providing 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. So, by using a BSP that supports monolithic methods, and requesting that BSP to be linked to those BioAPI_Units for the BFPs of other product providers, once that link is established, then the application can call those monolithic methods without taking into account that the functionality is provided by different vendors.

Last, but not least, there is a concern about security in certain operations. As biometric data is sensible personal data, some clients may require that the biometric application will not be able to access directly the user biometric data (i.e. the BIRs), avoiding the possibility of malware to tamper with such data. By using BFPs, all sensible data will be handled at the BSP level, and no BIR may be accessible to the application, not only simplifying the application development (by the use of monolithic methods), but also the security level achieved.

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 attach one of those BioAPI_Units of the BFP. This is done through a component registry and the procedure is further described in 6.4.

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 attachment 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 the following figure.

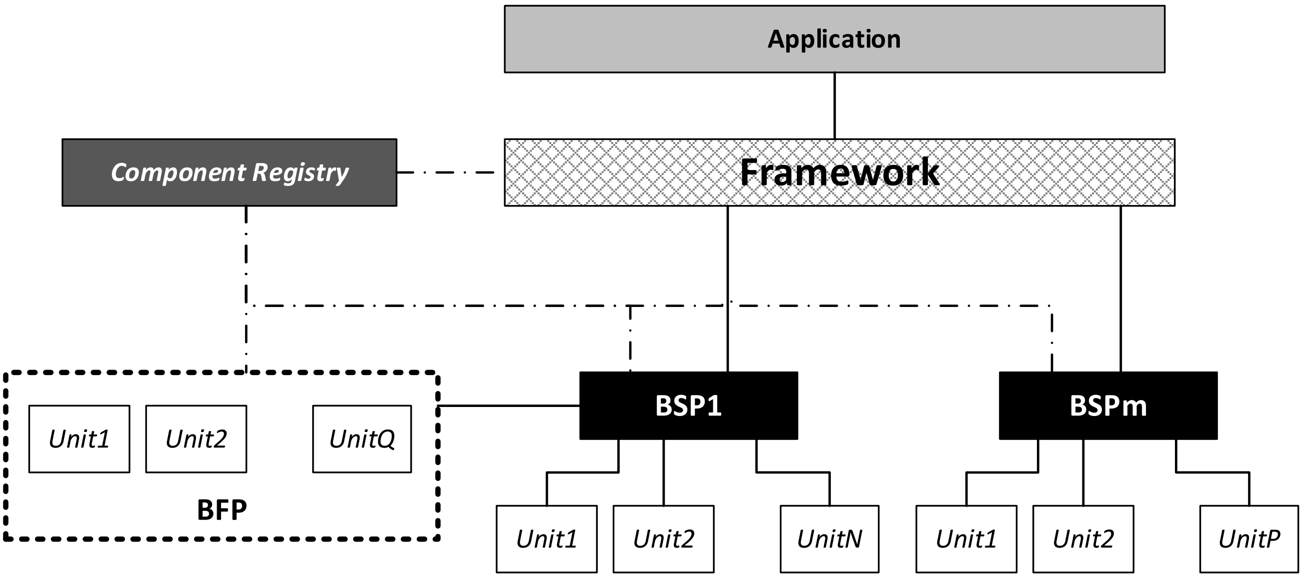

Figure 1: Generic structure of a framework-based application

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 this International Standards. The correspondence between both International Standards is given in the following table:

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 all parts of this International Standard, the Unit level will be also specified, but 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.

6.2 BioAPI compatibility requirements

This International Standard 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 compliante 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.

6.3 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 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 & 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 that, in order to avoid the re-programming of 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 designed for each step where the BSP requires to interact with the user, and for each of the processes to be called, the application provides the BSP with the selected functions, so that the BSP can call them within the execution of the process.

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. This 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.

6.4 Implementation guidelines

6.4.1 Basic concepts

The most common case of implementation is that one where the operating system has the Framework integrated, so that BSPs and BFPs developers can generate their own products and offer them for being installed into the Framework. Also, application developers do not have to even obtain the BSPs or BFPs, just only asking the Framework which BSPs and BFPs are installed, analyse their schemas, either automatically or allowing the user to select the ones he will like, and then perform the execution of the whole code by only performing calls to the Framework. Therefore it will be the application developer the one to specify the end-user about which kind of BSPs and/or BFPs will be required for the application and then it will have to be the end-user the one acquiring his/her desired OO BioAPI components.

In this kind of implementations, three kinds of developers are involved in the use of this specification:

- Those developers providing services to be accessed by applications (i.e. BSPs or BFPs). For this kind of developers, clauses 6.4.2, 6.4.3 and 6.4.4 are applicable.

- Those developers having to create the framework and integrate it into the system, so that it is accessible for both, the BSPs and BFP products, and the final applications. For this kind of developers, clause 6.4.5 is applicable.

- Those developers programing the final application and having to use some third party BSPs and/or BSPs. For this kind of developers, clause 6.4.6 is applicable.

6.4.2 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.

6.4.3 BFP development

As it was mentioned in 6.1, 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 a BSP can attach one of such BioAPI_Units, either by its own decision, or by request of the application. This relationship is implemented in the following way:

1) The application loads a BSP by using the BSPLoad method

2) When executing the BSPLoad() method, it is recommended that the BSP explores which BFPs are included in the Component Registry, and which BioAPI_Units does each of the BFPs contains. This is performed by using the callback function named BFPEnumerationCallback(), which obtains the BFPSchemas.

   i) For each of these BioAPI_Units, the BSP reads their UnitSchema structure to discover whether that BioAPI_Unit is compatible with the BSP, and therefore a supported BioAPI_Unit.

   ii) The BSP keeps an internal register of the supported BioAPI_Units and the BFP to which each of them belongs. It also assigns for each of the supported BioAPI_Units a unique identifier, called UnitID.

3) After the loading of the BSP, the application may ask the BSP about the supported BioAPI_Units, in order to discover which BioAPI_Units are available. This is done with the QueryUnits() method.

4) Before starting the execution of biometric functionality, the application shall attach a session for the selected BSP. This is done by using the BSPAttach() method, and it can be executed in two different ways, for each of the four BioAPI_Unit categories (i.e. archive, comparison, processing and sensor):

   i) The application can attach the session without telling the BSP which BioAPI_Unit to use for that category. This is done by stating DontCare in the relevant parameter of BSPAttach().

      I) Then the BSP can behave in the way it has been programmed to do. For example, if the BSP already contains a BioAPI_Unit of that category, it can decide to use its own BioAPI_Unit, forgetting about any other BioAPI_Units of that category in other BFPs, even if they are compatible. Or it can be programmed in a way to choose its considered best option among the BioAPI_Units available for that category.

      II) In case the BSP decides to use a BioAPI_Unit of that category that is not implemented in its own BSP (either because the BSP does not have such kind of BioAPI_Unit, or because it has decided any other option), the BSP shall firstly detach the current BioAPI_Unit (e.g. its own BioAPI_Unit), and then attach the one at the BFP by calling the UnitAttach() method for that BFP.

   ii) The application can decide to attach the session indicating the BioAPI_Unit selected for a certain category (or for all), by filling in the UnitIDs of those BioAPI_Units selected in each of the relevant parameters. In that case:

      I) If the BSP does not have a current BioAPI_Unit attached for that category, then it will have only to attach the requested BioAPI_Unit, by using the UnitAttach() method.