INTERNATIONAL STANDARD

ISO/IEC 24708

First edition
2008-12-15

Information technology — Biometrics — BioAPI Interworking Protocol

Technologies de l'information — Biométrie — Protocole d'interfonctionnement BioAPI

Reference number
ISO/IEC 24708:2008(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published by ISO in 2009

Published in Switzerland

CONTENTS

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.

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

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national 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 and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 24708 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 37, *Biometrics*, in collaboration with ITU-T. The identical text is published as ITU-T Rec. X.1083.

Introduction

This Recommendation | International Standard, the BioAPI interworking protocol (BIP), specifies the syntax, semantics, and encodings of a set of messages ("BIP messages") that enable a BioAPI-conforming application to request biometric operations in BioAPI-conforming biometric service providers (BSPs) across node or process boundaries, and to be notified of events originating in those remote BSPs. It also specifies extensions to the architecture and behaviour of the BioAPI framework (specified in ISO/IEC 19784-1) that support the creation, processing, sending, and reception of BIP messages.

A scenario where this Recommendation | International Standard would be applicable is where a national government decides to establish a system of biometric enrolment and authentication that will involve a central database of all enrolled persons in the country, with access to that database from biometric devices in health-care, social services, immigration, and security services. This is one of several applications where the BIP would be of use.

The BIP protocol is designed so that a conforming implementation does not have to support the whole functionality of a BioAPI framework. Several conformance classes are defined in this Recommendation | International Standard to accommodate various degrees of support of such functionality. This makes it possible to create lightweight implementations of this Recommendation | International Standard in which support of BioAPI-conforming applications or BioAPI conforming BSPs is either not possible or not required.

This Recommendation | International Standard uses the ASN.1 notation (see ITU-T Rec. X.680 series | ISO/IEC 8824-1 multi-part standard) to specify the protocol messages.

Clauses 7 to 11 contain informative overview material. Clauses 12 onwards (and some annexes) provide the normative specification.

Clause 7 describes the architecture of BIP.

Clause 8 describes the mechanism of remote GUI event notifications.

Clause 9 presents some examples of possible system configurations using BIP.

Clause 10 describes the format of the biometric data transferred by BIP.

Clause 11 describes the identification of BIP endpoints, applications, and BSPs.

Clause 12 provides an overview of BIP message exchanges.

Clause 13 contains general provisions which are invoked by other clauses.

Clause 14 specifies the general syntax of a BIP message.

Clause 15 specifies the mapping between BioAPI types and the corresponding ASN.1 types that occur as components of BIP messages.

Clause 16 specifies the syntax of some individual BIP messages and the actions to be performed when receiving a BioAPI function call or a BIP message related to a BioAPI function call.

Clause 17 specifies the syntax of some individual BIP messages, and the actions to be performed when receiving a BioAPI callback or a BIP message related to a BioAPI callback.

Clause 18 specifies a number of conceptual tables to be used by an implementation.

Clauses 19 to 33 contain specific provisions which are invoked by other clauses.

Annex A is normative and specifies the TCP/IP binding of BIP.

Annex B is normative and specifies additional provisions for the TCP/IP binding of BIP.

Annex C is normative and specifies the SOAP/HTTP binding of BIP.

Annex D is informative and clarifies the minimal requirements for simple systems.

Annex E is informative and provides examples of scenarios in which the BIP might be employed.

Annex F is normative and contains the complete ASN.1 specification of BIP.

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – Biometrics – BioAPI interworking protocol

1 Scope

1.1 This Recommendation | International Standard specifies the syntax, semantics, and encodings of a set of messages ("BIP messages") that enable a BioAPI-conforming application to request biometric operations in BioAPI-conforming biometric service providers (BSPs) across node or process boundaries, and to be notified of events originating in those remote BSPs.

NOTE – Both the local and the remote node or process can contain BSPs that provide storage and retrieval of biometric information records, processing or comparison of such records, or capture of biometric samples from one or more biometric sensors. It is possible for an individual node or process to contain both (one or more) applications that access remote BSPs, and (one or more) BSPs that are accessed by remote applications.

1.2 This Recommendation | International Standard also specifies extensions to the architecture and behaviour of the BioAPI framework that support the creation, processing, sending, and reception of BIP messages. A BioAPI framework conforming to this Recommendation | International Standard (a "BIP-enabled framework") creates, processes, sends, and receives BIP messages in close relationship with BioAPI function calls and callbacks. Outgoing BIP messages can be generated and sent by the framework as part of the handling of an incoming call or callback. Incoming BIP messages can cause actions to be performed by the framework as though a call or callback has been received.

1.3 This Recommendation | International Standard explicitly allows for BIP messages to be created, processed, sent, and received by a software entity (a "generic BIP entity") that is not necessarily a BIP-enabled framework.

NOTE – This makes it possible to create lightweight implementations of this Recommendation | International Standard in which support for BioAPI-conforming applications or BioAPI-conforming BSPs is either not possible or not required. There is no externally observable difference between the BIP messages created and sent by a generic BIP entity and those created and sent by a BIP-enabled framework. However, while a BIP-enabled framework is required to fully and properly implement the relationship specified herein between BIP messages and BioAPI function calls or callbacks, a generic BIP entity has no such obligation (see clause 6).

1.4 This Recommendation | International Standard specifies the use of any of several commonly available transport protocols for the transfer of BIP messages between a pair of software entities ("BIP endpoints").

1.5 Standardization of biometric data blocks (carrying raw, intermediate, or processed biometric samples) is not in the scope of this Recommendation | International Standard.

NOTE – Standardization of such formats is performed by the various parts of ISO/IEC 19794.

1.6 Standardization of biometric information records (each containing one or more biometric data blocks together with identifying and other meta-information) is not in the scope of this Recommendation | International Standard.

NOTE – Standardization of the elements of such formats is performed by ISO/IEC 19785-1, which also contains the specification of a number of standardized biometric information record formats.

1.7 Comparison algorithms for biometric identification or verification are not in the scope of this Recommendation | International Standard.

1.8 The definition of security mechanisms is not in the scope of this Recommendation | International Standard, but a number of bindings to secure transport protocols are specified in order to support secure exchanges between BIP endpoints.

1.9 The classification of, determination of, or requirements on the performance of biometric systems is not in the scope of this Recommendation | International Standard.

1.10 This Recommendation | International Standard specifies a Version 1 of the BioAPI interworking protocol (BIP), and assigns it the ASN.1 object identifier value {iso standard 24708 version (1)} (see ITU-T Rec. X.680 | ISO/IEC 8824-1 for the meaning of this notation).

1.11 ISO/IEC 19784-1 specifies version 2.0 and 2.1 of BioAPI. This Recommendation | International Standard provides support for only version 2.1.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.667 (2004) | ISO/IEC 9834-8:2005, *Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components*.
- ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.
- ITU-T Recommendation X.681 (2002) | ISO/IEC 8824-2:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.
- ITU-T Recommendation X.682 (2002) | ISO/IEC 8824-3:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification*.
- ITU-T Recommendation X.683 (2002) | ISO/IEC 8824-4:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.
- ITU-T Recommendation X.691 (2002) | ISO/IEC 8825-2:2002, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*.
- ITU-T Recommendation X.693 (2001) | ISO/IEC 8825-4:2002, *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER)*.
- ITU-T Recommendation X.693 (2001)/Amd.1 (2003) | ISO/IEC 8825-4:2002/Amd.1:2004, *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER) – Amendment 1: XER encoding instructions and EXTENDED-XER*.

2.2 Paired Recommendations | International Standards equivalent in technical content

None.

2.3 Additional references

- ISO/IEC TR 8802-1:2001, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 1 : Overview of Local Area Network Standards*.
- 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 19785-3:2007, *Information technology – Common Biometric Exchange Formats Framework – Part 3: Patron format specifications*.
- ISO/IEC 19794 (all parts), *Information technology – Biometric data interchange formats*.
- IETF RFC 768 (1980), *User Datagram Protocol*.
- IETF RFC 791 (1981), *Internet Protocol*.
- IETF RFC 793 (1981), *Transmission Control Protocol*.
- IETF RFC 826 (1982), *Ethernet Address Resolution Protocol*.
- IETF RFC 1945 (1996), *Hypertext Transfer Protocol – HTTP/1.0*.
- IETF RFC 2131 (1997), *Dynamic Host Configuration Protocol*.
- IETF RFC 2136 (1997), *Dynamic Updates in the Domain Name System (DNS UPDATE)*.
- IETF RFC 2462 (1998), *IPv6 Stateless Address Autoconfiguration*.

- IETF RFC 2616 (1999), *Hypertext Transfer Protocol – HTTP/1.1*.
- IETF RFC 2818 (2000), *HTTP Over TLS*.
- IETF RFC 3315 (2003), *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*.
- IETF RFC 3927 (2005), *Dynamic Configuration of IPv4 Link-Local Addresses*.
- IETF RFC 3987 (2005), *Internationalized Resource Identifiers (IRIs)*.
- IETF RFC 4443 (2006), *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*.
- W3C SOAP 1.2:2007, *SOAP Version 1.2*.
- W3C SOAP MTOM:2005, *SOAP Message Transmission Optimization Mechanism*.
- W3C XMLENC:2002, *XML Encryption Syntax and Processing*.
- W3C XMLDSIG:2002, *XML – Signature Syntax and Processing*.

3 Conformance

The terms and definitions specified in ISO/IEC 19784-1 and ISO/IEC 19785-1 also apply to this Recommendation | International Standard.

The following additional terms and definitions are used:

3.1 acknowledgement BIP message: BIP message conveying a response (acknowledgement) to a prior notification BIP message.

NOTE – Not all notification BIP messages have a corresponding acknowledgement BIP message.

3.2 BIP-enabled framework: Augmented version of the BioAPI framework that is capable of creating, processing, sending, and receiving BIP messages in close relationship with BioAPI function calls and callbacks.

NOTE – Not all implementations of the BIP-enabled framework are BioAPI frameworks fully conforming to ISO/IEC 19784-1 (see the Note to 6.7).

3.3 BIP endpoint: Runtime entity, identified by an endpoint IRI, capable of sending and receiving BIP messages, and containing either a running generic BIP entity, or a running BIP-enabled framework with one component registry, zero or one running BioAPI application, and zero or more running BSPs.

3.4 BIP link: Logical connection between two BIP endpoints, consisting of a mandatory request/response link channel and an optional notification/acknowledgement link channel, in which one BIP endpoint plays the role of master and the other plays the role of slave.

NOTE – At most one BIP link can exist between any pair of BIP endpoints for either assignment of roles. If a BIP link has two link channels, one of the two BIP endpoints will be the master endpoint in both link channels, and the other one will be the slave endpoint in both. A BIP link cannot exist between a pair of BIP endpoints if both are master-role-capable or both are slave-role-capable.

3.5 BIP message: Message that can be sent from a BIP endpoint to another BIP endpoint through a link channel.

3.6 BSP access UUID: Non-persistent UUID that is dynamically generated and assigned by a BIP-enabled framework to a BSP that is available for loading into a given BIP endpoint (either the local one or a remote one), and which unambiguously identifies both the BSP and (implicitly) the BIP endpoint.

NOTE – BSP access UUIDs are meaningless outside the BIP endpoint in which they have been generated.

3.7 BSP product UUID: Persistent UUID that has been assigned to a BSP software product by its vendor.

NOTE 1 – The BSP product UUID is expected to remain the same across multiple installations of the same BSP on different systems.

NOTE 2 – A BSP product UUID can be generated and registered at http://www.itu.int/ITU-T/asn1/uuid.html (see ITU-T Rec. X.667 | ISO/IEC 9834-8).

3.8 CBEFF: Data elements and BIR formats specified in ISO/IEC 19785-1.

3.9 endpoint IRI: IRI that unambiguously identifies a BIP endpoint.

NOTE – There are no constraints on the form of this IRI. In general (unless a binding specification prescribes otherwise), there is no relationship between the IRI scheme of the endpoint IRI and the binding(s) supported by the underlying implementation. This implies that, in general, an endpoint IRI may not provide, by itself, sufficient information for locating the BIP endpoint on a network.

3.10 generic BIP entity: Software entity that is capable of creating, processing, sending, and receiving BIP messages, but which does not necessarily implement the BioAPI API and the whole functionality of a BioAPI framework (including access to the component registry and to local BSPs).

3.11 link channel: Logical connection from one BIP endpoint to another BIP endpoint using a specific transport protocol binding, in which one BIP endpoint plays the role of master and the other plays the role of slave.

3.12 master endpoint (of a given BIP endpoint): BIP endpoint that has been logically connected with a BIP link to the given (slave) BIP endpoint, and which is capable of sending request BIP messages to the given BIP endpoint and processing notification BIP messages received from it.

NOTE – A given BIP endpoint may play both the role of master and the role of slave at the same time, with any number of other BIP endpoints, provided that an appropriate set of BIP links are established.

3.13 notification BIP message: BIP message conveying the notification of an event of interest to the receiving BIP endpoint.

3.14 notification/acknowledgement link channel: Link channel through which notification and acknowledgement BIP messages are transferred.

NOTE – A notification/acknowledgement link channel can only exist as part of a BIP link. Its presence in a BIP link is optional.

3.15 remote access policy: Determination of which local BSPs, which local BFPs, and which BioAPI units (managed by those BSPs and BFPs) a BIP endpoint makes available for use by another BIP endpoint.

3.16 request BIP message: BIP message conveying a request for an action to be performed by the receiving BIP endpoint.

3.17 request/response link channel: Link channel through which request and response BIP messages are transferred and which is always present in a BIP link (see 3.4).

NOTE – A request/response link channel can only exist as part of a BIP link.

3.18 response BIP message: BIP message conveying a response to a prior request BIP message.

3.19 slave endpoint (of a given BIP endpoint): BIP endpoint to which the given (master) BIP endpoint has been logically connected with a BIP link, and which is capable of processing request BIP messages received from the given BIP endpoint and sending notification BIP messages to it.

NOTE – A given BIP endpoint may play both the role of master and the role of slave at the same time, with any number of other BIP endpoints, provided that an appropriate set of BIP links are established.

3.20 transport protocol binding: Physical realization of a link channel, specifying what transport protocol to use, how to encode BIP messages, how to compose transport-level messages carrying the encoded BIP messages, and other details about the usage of the transport protocol.

4 Abbreviations

For the purpose of this Recommendation | International Standard, the following abbreviations apply.

ARP	Address Resolution Protocol
ASN.1	Abstract Syntax Notation One
BDB	Biometric Data Block
BFP	Biometric Function Providers
BIP	BioAPI Interworking Protocol
BIR	Biometric Information Record
BSP	Biometric Service Providers
DHCP	Dynamic Host Configuration Protocol
FPI	Function Provider Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer
IP	Internet Protocol (see IETF RFC 791)
IRI	Internationalized Resource Identifier
MAC	Media Access Control
MTOM	Message Transmission Optimization Mechanism

PER	Packed Encoding Rules
SOAP	Simple Object Access Protocol
SPI	Service Provider Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UUID	Universally Unique Identifier

5 Conventions

The following typographic and colour conventions have been followed in this Recommendation | International Standard to facilitate reading even though they are not essential to understand the Recommendation:

- C language names and definitions (such as BioAPI type definitions, BioAPI function definitions, and input parameters of BioAPI functions) are displayed as in the following example: **BioAPI_Init**; output parameters of BioAPI functions are printed as in the following example: **Response**;
- ASN.1 language names and definitions (such as ASN.1 type definitions, ASN.1 component names, and BIP message types) are displayed as in the following example: **masterEndpointIRI**;
- XML names and documents (such as XML Schema definitions, global element names, and examples of XML-encoded BIP messages) are displayed as in the following example: **<bip:verifyMatch/>**.

6 Conformance

6.1 Clauses 7 to 33 specify the syntax and semantics of BIP messages and the behaviour of a BIP-enabled framework on reception of incoming BioAPI function calls (made by the local application), callbacks (made by a local BSP), and BIP messages.

6.2 A BIP-enabled framework is one type of implementation of this Recommendation | International Standard. The other type of implementation is a generic BIP entity. A generic BIP entity has the ability to create, process, send, and receive BIP messages, but does not necessarily implement the BioAPI API and the whole functionality of a BioAPI framework (including access to the component registry and to local BSPs) internally.

6.3 Conformance to this Recommendation | International Standard is organized into two conformance levels (level 1 and level 2) and three role capability classes (master-role-capable, slave-role-capable, and dual-role-capable). This gives a total of six conformance classes as follows:

Table 1 – Conformance classes

		Conformance level	
		level 1 (generic BIP entity)	**level 2 (BIP-enabled framework)**
Role capability class	**master-role-capable**	*master-role-capable generic BIP entity*	*master-role-capable BIP-enabled framework*
	slave-role-capable	*slave-role-capable generic BIP entity*	*slave-role-capable BIP-enabled framework*
	dual-role-capable	*dual-role-capable generic BIP entity*	*dual-role-capable BIP-enabled framework*

6.4 The provisions of clauses 7 to 33 shall be applied differently for each of the two conformance levels, as follows:

a) for a BIP-enabled framework, they shall be applied as specified;

b) for a generic BIP entity, they shall be interpreted as referring to an ideal framework conceptually present within the generic BIP entity, with no presumption that the ideal BioAPI and BioAPI SPIs of the ideal framework are visible to any external observer or test.

NOTE – The internal structure of a generic BIP entity is not merely invisible. It is also totally unconstrained, and is not specified for the purposes of conformance or conformance testing.

6.5 A software entity may claim to be a master-role-capable BIP-enabled framework if and only if it:

a) exposes a BioAPI API to a local application;

b) processes incoming BioAPI calls from a local application as specified in clauses 7 to 33;

c) processes incoming notification BIP messages as specified in those clauses; and

d) never produces BIP messages except as specified in those clauses.

6.6 A software entity may claim to be a slave-role-capable BIP-enabled framework if and only if it:

a) uses the BioAPI SPI exposed by local BioAPI-conforming BSPs to interact with them;

b) processes incoming callbacks from local BSPs as specified in clauses 7 to 33;

c) processes incoming request BIP messages as specified in those clauses; and

d) never produces BIP messages except as specified in those clauses.

6.7 A software entity may claim to be a dual-role-capable BIP-enabled framework if and only if it satisfies the conditions for both a master-role-capable and a slave-role-capable BIP-enabled framework.

NOTE – A dual-role-capable BIP-enabled framework is the only type of implementation of BIP that is required to support *both* the BioAPI API *and* the use of local BSPs, and therefore it is the only one that can be regarded as a true superset of the BioAPI framework as specified in ISO/IEC 19784-1.

6.8 A software entity may claim to be a generic BIP entity (master-role-capable, slave-role-capable, or dual-role-capable) if and only if it is possible to determine, for any possible sequence (of arbitrary length) of incoming and outgoing BIP messages, a congruous sequence of actions which could have been performed by an ideal BIP-enabled framework (of that role capability class) conceptually present within the software entity and that would have resulted in that sequence of BIP messages.

NOTE – Clauses 7 to 33 are worded in such a way to ensure that a BIP implementation does not normally need to know the conformance level of another BIP implementation with which it will exchange BIP messages. The other implementation will either be a conforming BIP-enabled framework or (if it is a generic BIP entity) it will behave as though it contained a BIP-enabled framework. For example, it is not possible, just by observing the BIP messages coming from the other implementation, to determine whether that implementation uses a real BioAPI-conforming BSP to perform biometric operations or uses some non-standard internal architecture.

6.9 As a corollary of the previous subclauses, the following interfaces are subject to conformance testing for each conformance class:

Table 2 – Conformance testing

Conformance class	BioAPI API	BioSPI API	Messaging interface
master-role-capable generic BIP entity			X
slave-role-capable generic BIP entity			X
dual-role-capable generic BIP entity			X
master-role-capable BIP-enabled framework	X		X
slave-role-capable BIP-enabled framework		X	X
dual-role-capable BIP-enabled framework	X	X	X

6.10 Conformance to the transport protocol bindings specified in Annexes A and B is defined separately for each annex. However, conformance testing of an implementation cannot be performed unless there is at least one binding that is supported both by the implementation and by the testing tool. In addition, conformance with respect to the correct processing of notification BIP messages cannot be assessed unless the binding used in the test supports the transfer of notification BIP messages.

6.11 An implementation claiming conformance to Annex A does not need to also claim conformance to Annex B. However, conformance to Annex B requires conformance to Annex A.

6.12 Conforming implementations are additionally required to detect and to process bit-patterns in incoming messages that do not form part of the BIP protocol (or that represent correct messages that are not allowed by previous exchanges) in a manner that prevents denial of service attacks by an outside source, or by a Trojan horse inside an otherwise conforming system. This requirement applies to all implementations claiming conformance to this Recommendation | International Standard, even if the implementation is intended for use on a physically secured network. It is not optional if conformance is claimed.

NOTE – This requirement essentially means that a conforming receiver will make no assumptions on the correctness of encodings and messages that are received, even on an established connection from an authenticated BIP sender. Conforming implementations should not "crash" due to (for example) buffer over-run problems. It is expected that if BIP becomes widely used, extensive test-suites will be produced to identify incorrect implementations that result in vulnerability to denial of service attacks arising from messages that are not correct BIP messages, whether coming from an authenticated sender or not.

7 BIP architecture

At the heart of BIP are the concepts of BIP-enabled framework, BIP message, BIP endpoint, BIP link, master/slave endpoints, and transport protocol binding, in addition to BioAPI concepts such as BSP and local application.

7.1 BIP-enabled frameworks

7.1.1 One of the purposes of BIP is to enable a BioAPI application to use a remote BSP in the same ways as it would use a local BSP. This is a very concise statement that is clarified in the following subclauses.

7.1.2 The distinction between local and remote BSPs is based on the distinction between local and remote BioAPI frameworks, which is as follows. For a given running BioAPI application, a local BioAPI framework is a running instance of a BioAPI framework product that exposes its BioAPI API to the application, whereas a remote BioAPI framework is any other running instance of a BioAPI framework product (which may be located either in a different computer or in a different process within the same computer). A local BSP is a loadable or running BSP whose schema is present in the component registry of a local BioAPI framework, and which is therefore accessible to the application through the BioAPI API of that local BioAPI framework. A remote BSP is a loadable or running BSP whose schema is present in the component registry of a remote BioAPI framework but not in the component registry of any local BioAPI framework, and which is therefore not accessible to the application through the BioAPI API of any local BioAPI framework.

7.1.3 There are many situations in which a BioAPI application may wish to access a remote BSP. Some examples are:

- a sensor device may be physically attached to a computer different from the one in which the application is running, and be managed by a BSP that is loaded into the computer to which the sensor device is attached;
- a template database may be located on a computer different from the one in which the application is running, and be managed by a BSP that is loaded into the computer where the template database is located;
- multiple registered copies of a BSP product implementing a comparison algorithm may be made available for execution on multiple servers in order to optimize the use of computing resources;
- a watch-list search service may be provided using a BSP running on a server;
- on a given computer there may be a security policy restricting access to a template database, so that the BSP that manages the template database and an application that accesses that BSP have to be given different access control permissions, and therefore cannot run in the same process, even though they are allowed to run in the same computer;
- executing an application in a separate process from that in which a BSP is running may help increase the reliability of the whole system (when the application crashes, the BSP can remain available, and vice versa);
- executing an application in a separate process from that in which a BSP is running enables multiple concurrent applications to share the same running instance of the BSP, which may help the BSP to optimize its internal resource management.

7.1.4 Access to remote BSPs through the BioAPI API of a local framework is not supported by BioAPI but is supported by BIP.

7.1.5 The BioAPI API exposed by a BIP-enabled framework is syntactically identical to the one exposed by a BioAPI framework (the signatures of all functions are the same), but the semantics of most BioAPI functions is extended and specialized. When an application uses the BioAPI API of a BIP-enabled framework, the same sequences of BioAPI calls and callbacks work for both local and remote BSPs. It would always be possible to replace a BioAPI framework with a BIP-enabled framework with no visible changes in behaviour (assuming that all currently registered BSPs are re-registered in the new framework).

7.1.6 The above implies that there is no fundamental difference between a BioAPI application (not BIP-aware) that only uses local BSPs and one (BIP-aware) that uses both local and remote BSPs, apart from the portions of the application that deal specifically with the location of the BSPs (if any). Both local and remote BSPs are listed in the array returned by **BioAPI_EnumBSPs**; both local and remote BSPs are "loaded" by calling **BioAPI_BSPLoad** and are "attached" by calling **BioAPI_BSPAttach**; both local and remote BSPs can send unit event notifications and GUI event notifications to the application; and so on. A BioAPI application will still work even if the BioAPI framework is replaced with a BIP-enabled framework and some or all of the local BSPs are moved to a remote BIP-enabled framework (and thus become remote BSPs).

7.2 BIP messages

7.2.1 This Recommendation | International Standard specifies a set of messages that can be exchanged between two BIP-enabled frameworks (using transport protocol bindings) and precise rules for creating and processing those messages in relation with BioAPI function calls and callbacks. A BIP-enabled framework is capable of creating, processing, sending, and receiving BIP messages. The logical connection between two running BIP-enabled frameworks which supports the exchange of BIP messages is called a BIP link.

7.2.2 The messaging protocol of BIP is modelled upon the BioAPI API and is closely related to it. Most BioAPI functions and callbacks have a corresponding pair of BIP messages (a request message and a response message, or a notification message and an acknowledgement message), although there are exceptions.

7.2.3 The abstract message syntax of BIP is specified in ASN.1 notation. At the top level, there is a choice between four main kinds of messages (request, response, notification, and acknowledgement). The BIP messaging protocol does not specify the use of any particular set of encoding rules for the ASN.1 type definitions, because the transfer of BIP messages is modelled as a conceptual transfer of abstract values (see 7.4). Encoding rules are specified in the individual transport protocol bindings (see 7.5).

7.2.4 All four kinds of BIP messages carry a link number, which identifies messages exchanged across a BIP link (see 7.4), and an identifier (positive integer), which is either a request identifier (for requests and responses) or a notification identifier (for notifications and acknowledgements). Request and notification BIP messages sent across each BIP link are independently numbered, starting from an arbitrarily chosen number, and are incremented on each message being sent. When an identifier reaches the top of the allowed range (4294967295), it restarts from zero. A response BIP message is required to carry the same link number and identifier as the corresponding request BIP message. The purpose of the link number and identifier in BIP is to facilitate the pairing of responses with the corresponding requests and the pairing of acknowledgements with the corresponding notifications. Identifiers cannot be relied upon for determining the order of transmission of BIP messages.

7.3 BIP endpoints

7.3.1 Although it would be possible to specify the exchange of BIP messages in terms of a local and a remote BIP-enabled frameworks communicating over a BIP link, this Recommendation | International Standard adopts a slightly different model based on the concept of BIP endpoint, which accommodates implementations that are BIP-enabled frameworks as well as implementations that are not frameworks, as described in the following subclauses.

7.3.2 A BIP endpoint is a conceptual runtime software entity, which is identified by a unique IRI and is capable of creating, processing, sending, and receiving BIP messages. A BIP endpoint sends BIP messages to another BIP endpoint through a BIP link, which is a logical connection established between two BIP endpoints. One of the two BIP endpoints in a BIP link plays the role of master endpoint with respect to that link, the other plays the role of slave endpoint with respect to that link.

7.3.3 A BIP endpoint may have an internal structure consisting of the following runtime components:

a) one running instance of a BIP-enabled framework (which is responsible for sending and receiving the BIP messages that appear to be sent and received by the BIP endpoint);

b) one BioAPI component registry (managed by the framework in a));

c) zero or one running instance of a BioAPI application (which uses the BioAPI API of the framework in a));

d) zero or more running instances of BSPs (whose BioAPI SPI is used by the framework in a)); and

e) zero or more running instances of BFPs (whose BioAPI FPI is used by some of the BSPs in d)).

7.3.4 Alternatively, a BIP endpoint may have a different internal structure or an unspecified internal structure, in which case it is said to contain a generic BIP entity (a different conformance class from a BIP-enabled framework). However, the two BIP endpoints in a BIP link have no visibility of each other's internal structure. All the provisions of this Recommendation | International Standard regarding the exchange of BIP messages are expressed in terms of a framework sending a message to a (master or slave) BIP endpoint or receiving a message from a (master or slave) BIP endpoint, and never refer to the BIP-enabled framework contained in that BIP endpoint (which may or may not contain a BIP-enabled framework, depending on its conformance class).

7.3.5 There are many situations in which an implementer may wish to create a generic BIP entity instead of a BIP-enabled framework. Some examples are:

– computing platforms with constrained resources (such as embedded systems), which are built to perform a very specific set of biometric functions and do not need to support arbitrary BSPs or arbitrary applications, but need to support the exchange of BIP messages;

- implementations of BIP that wish to expose an API different from the standard BioAPI 2.0 defined in ISO/IEC 19784-1 API or wish to use biometric service modules different from standard BioAPI 2.0 BSPs;
- implementations of BIP that wish to support the same functionalities of a BioAPI framework while providing an API in a programming language other than C;
- proxies, gateways, firewalls, or other intermediaries which may perform a variety of functions (including translation between different transport protocol bindings, message routing, message logging, security checks, load balancing, etc.); these functions normally result in another BIP message of the same type being sent to another BIP endpoint, instead of a biometric operation being performed by that endpoint.

7.3.6 As stated above, the internal structure of a BIP endpoint (and so whether the endpoint contains a BIP-enabled framework or an embedded implementation, a non-standard framework, a proxy, a gateway, etc.) is not visible to any other BIP endpoint that is engaged in a BIP link with it. The BIP endpoint will exhibit the same behaviour in all cases as far as can be observed from the BIP messages being exchanged.

7.3.7 The limits imposed on the number of BioAPI applications within a BIP endpoint (at most one) and on the number of endpoint IRIs (one) do not prevent the existence of implementations of this Recommendation | International Standard in which a BIP-enabled framework implementation supports multiple BioAPI applications and multiple endpoint IRIs simultaneously. Such an implementation can be modelled as a set of conceptual BIP endpoints, each having exactly one endpoint IRI and containing at most one BioAPI application. The provisions of this Recommendation | International Standard then apply to each one of those conceptual BIP endpoints.

7.4 BIP links

7.4.1 A BIP link is a logical connection established between two BIP endpoints for the exchange of BIP messages. This is a very abstract concept. There are many ways to physically realize a BIP link.

7.4.2 The term "send", as used in this Recommendation | International Standard, means the conceptual transfer of a BIP message from a BIP endpoint (sender) to the BIP link between the sending endpoint and the receiving endpoint. The message being transferred is an abstract value of type **BIPMessage** (see clause 14), which is placed on the link as a result of the send. The term "receive", as used in this Recommendation | International Standard, means the conceptual transfer of a BIP message from the BIP link to the receiving BIP endpoint. The message being transferred is removed from the link as a result of the receive. No assumptions are made at this level about the physical nature of the BIP link, about the underlying transport protocol, about the encoding of the BIP message being transferred, and about time-related or space-related aspects of sending or receiving (such as waiting and queuing).

7.4.3 The roles of master and slave with respect to a given BIP link constrain the types of BIP messages that a BIP endpoint can send through the link. A master endpoint can only send request BIP messages and acknowledgement BIP messages through the link, and a slave endpoint can only send response BIP messages and notification BIP messages through the link. A BIP link is always created by a BIP endpoint that intends to act as a master endpoint with respect to that link and knows in advance the endpoint IRI of the other BIP endpoint. A BIP endpoint that accepts to participate in a link being created by another BIP endpoint implicitly accepts to act as a slave endpoint with respect to that link, and may or may not know in advance the endpoint IRI of the other BIP endpoint (but will learn it as the link is established). There may be at most one BIP link between two BIP endpoints for either assignment of roles, but there is no limit to the number of links in which a BIP endpoint may be engaged (in either role). A BIP endpoint may refuse to participate in a link being created by a certain BIP endpoint, or may always refuse to act as a slave BIP endpoint.

7.4.4 Each request BIP message sent by the master endpoint through a BIP link is followed by a response BIP message sent by the slave endpoint through the same link. Some (but not all) notification BIP messages sent by the slave endpoint through a link are followed by an acknowledgement BIP endpoint sent by the master endpoint through the same link. Unsolicited responses (not preceded by a request) and unsolicited acknowledgements (not preceded by a notification) are not allowed. Examples of request BIP messages are those originating from a call to **BioAPI_BSPLoad** or **BioAPI_VerifyMatch** made by the local application within the master endpoint. Examples of notification BIP messages are those originating from a callback to the local framework's unit event handler or GUI event handler made by a local BSP within the slave endpoint.

7.4.5 Here is an example of a possible interaction between two linked BIP endpoints that both contain a BIP-enabled framework. A BioAPI call made by the application in the master endpoint and directed to a BSP in the slave endpoint will normally result in a request BIP message being sent by the master framework to the slave framework. The slave framework will process the message as if it had received a function call on its BioAPI API, and after the processing, it will send a response BIP message to the master framework, which will then return control to the local application. A notification callback (such as a unit event notification or a GUI event notification) made by a BSP in the slave endpoint to its framework will normally result in a notification BIP message being sent by the slave framework to the master framework. The master framework will process the message by making a callback to a handler function

provided by the local application. For some notifications, the master framework will send an acknowledgement message to the slave framework, which will then return control to the local BSP. For other notifications, the slave framework will return control to the BSP as soon as the notification BIP message has been sent to the master framework.

7.5 Transport protocol bindings

7.5.1 A transport protocol binding is a physical realization of a link channel (either a request/response link channel or a notification/acknowledgement link channel). In most cases, a binding specification will reference a commonly available transport protocol rather than specify a new one. A binding specification has to provide complete details about:

- the transport protocol;
- selection of parameters and options of the transport protocol;
- bit-level encoding of BIP messages;
- addressing mechanisms or conventions;
- possible sharing of a transport-level connection among multiple abstract link channels;
- definition of transport-level messages that carry the encoded BIP messages;
- possible grouping of multiple encoded BIP messages into a single transport-level message;
- any additional mechanisms in support of security, reliability, routing, etc.

7.5.2 A BIP link has a request/response link channel, and may or may not have a notification/acknowledgement link channel. If a BIP link does not have a notification/acknowledgement link channel, then the slave endpoint cannot send notification BIP messages to the master endpoint. If a BIP link has both channels, the two channels may use either the same binding or different bindings. In most cases, a BIP link has both channels and the two channels use the same binding. However, the ability to use different bindings for the two channels (or avoid the notification/acknowledgement channel altogether) is useful in many situations where different costs and different benefits are associated with the use of a particular binding by each link channel between two given BIP endpoints.

7.5.3 A conforming implementation of this Recommendation | International Standard (of any conformance class) may use any transport protocol binding (either standardized or proprietary) for each link channel when establishing a BIP link with another BIP endpoint, provided that the other BIP endpoint supports those transport protocol bindings. Several standardized bindings are specified in Annexes A to C in order to facilitate interoperability, but there is no requirement for implementations to support any one of them.

7.6 Creation and destruction of BIP links

7.6.1 BIP links can be established by either:

a) managed connections; or

b) by automatically-established connections using the so-called Plug and Play (PnP) mechanisms.

NOTE 1 – Specific transport protocol bindings specified in the annexes define a default PnP mechanism for that transport protocol binding.

NOTE 2 – Specification of the mechanisms to be used to establish managed connections is out of the scope of this Recommendation | International Standard.

7.6.2 The PnP mechanisms specified in the annexes provide service discovery for BIP master endpoints. Service announcement mechanisms for slave endpoints are also specified, but their use is optional. The PnP mechanisms contain definitions on how to use generic security mechanisms. Specification of the security mechanisms to be used is not in the scope of this Recommendation | International Standard.

7.6.3 The protocol messages creating and destroying BIP links may be transmitted separately from the BIP messages, depending on the capabilities of the specific protocol binding.

EXAMPLE: TCP/IP provides an identification of types of messages between the same endpoints by using different port numbers for the different types of messages. For the TCP/IP binding of BIP, different ports are used for the PnP mechanism and for BIP messages.

7.6.4 A BIP link can be created by a BIP-enabled framework either implicitly or explicitly. A BIP link is explicitly created by the framework when the framework processes an incoming call to **BioAPI_LinkToEndpoint** made by the local application. The BIP endpoint identified by the endpoint IRI specified by the application becomes the slave endpoint in the newly created BIP link, and the local endpoint becomes the master endpoint. A BIP link is implicitly created by the framework whenever the framework chooses to do so while processing an incoming call to **BioAPI_Init** or **BioAPI_InitEndpoint**. In this case, the local application has no direct control over the creation of BIP links (the

framework chooses a set of BIP endpoints and attempts to create a link to each of them). Obviously, it is possible to use the two methods in combination. Normally, the framework chooses the binding(s) to use for each link channel, but the application can suggest a particular binding or particular binding parameters via an auxiliary parameter of the function **BioAPI_LinkToEndpoint** (the format of this auxiliary parameter is not standardized). A BIP link is destroyed when the framework processes an incoming call to **BioAPI_UnlinkFromEndpoint** (which specifies an endpoint IRI), or when it processes an incoming call to **BioAPI_Terminate** (all existing links are destroyed), or when it receives a **masterDeletionEvent** notification BIP message from a slave endpoint or sends a **masterDeletionEvent** notification BIP message to a master endpoint (the link with that BIP endpoint is destroyed), or when a failure of a transport-level connection is detected.

7.6.5 The creation of a BIP link occurs in two phases. In the first phase (optional), a transport-level connection is established for each channel (either one or two) of the new BIP link. The first phase may be skipped if it is possible to utilize a pre-existing transport-level connection or if there is no need to establish a permanent transport-level connection. At this point, the roles of master and slave are assigned to the two endpoints (the creator of the link becomes the master endpoint). In the second phase, the master framework sends an **addMaster** request BIP message to the slave endpoint. On reception of the **addMaster** request BIP message from the master, the slave framework will add the endpoint IRI of the new master to its internal **MasterEndpoints** table, and will return the following information in the **addMaster** response BIP message to the master: the framework schema, a list of BSP schemas, and a list of BFP schemas, copied from its local component registry. On reception of the **addMaster** response BIP message from the slave, the master framework will add the framework schema to its internal **VisibleEndpoints** table, the BSP schemas to its internal **VisibleBSPRegistrations** table, and the BFP schemas to its internal **VisibleBFPRegistrations** table. These three tables contain consolidated schema information coming from the component registries of all slave endpoints and from the local component registry, and are consulted (*in lieu* of the local component registry) by the functions **BioAPI_EnumFrameworks**, **BioAPI_EnumBSPs**, **BioAPI_EnumBFPs**, **BioAPI_BSPLoad**, **BioAPI_BSPAttach**, etc. These tables are updated not only when a link to a slave is created or destroyed, but also when a change occurs in the component registry of any slave endpoint or in the local component registry.

7.6.6 The destruction of a BIP link also occurs in two phases. In the first phase, a **deleteMaster** request BIP message received by a slave from a master will cause the deletion of the **MasterEndpoints** table entry corresponding to that master. Alternatively, if a slave is terminating (for any reason), it will send a **masterDeletionEvent** notification BIP message to all of its masters. In each case, the master will delete all the entries pertaining to that slave from its **VisibleEndpoints**, **VisibleBSPRegistrations**, and **VisibleBFPRegistrations** tables. In the second phase (optional), the transport-level connection(s) corresponding to the channel(s) of the link are destroyed by either BIP endpoint. The second phase may be skipped if each BIP endpoint wishes to be able to utilize the same transport-level connection(s) for a BIP link to be established in the future. In case a BIP endpoint detects the failure of a transport-level connection of a link, it will delete from its tables all the entries pertaining to the endpoint (either master or slave) associated with that link, and any pending function calls or callbacks related to one of those entries will return with an error.

7.6.7 Due to the way schema information is transferred from a slave endpoint to a master endpoint, a BIP link is neither symmetric nor transitive. For example, if an endpoint A links to an endpoint B, an application in A will "see" the BSPs registered in B, but an application in B will not see the BSPs registered in A, unless another (independent) link is established from B to A. If an endpoint A links to an endpoint B which links to an endpoint C, an application in A will "see" the BSPs registered in B, but not those registered in C, unless another (independent) link is established from A to C. (Note, however, that the non-transitivity statement only applies to implementations conforming to level-2, but not to level-1 implementations such as proxies and gateways acting as slaves, because conformance of those implementations is defined exclusively with regard to the sequences of BIP messages exchanged with a master endpoint. A proxy or gateway can in fact violate the transitivity rule, so long as it produces BIP messages that could have been produced by a BIP-enabled framework with a suitable component registry and a suitable set of local BSPs and BFPs.)

8 Remote GUI event notifications

8.1 BIP allows an application to run on a different computer from the one in which the BSP is loaded, which is usually the one where the sensor is physically attached. If a non-BIP-aware BioAPI application running in a BIP setup requests GUI event notification callbacks and the BSP supports those callbacks, the callbacks will be sent to the application. On reception of those callbacks, the application may display a GUI on the computer where it is running, but that computer may not be the computer with the sensor attached, which is the one that the subject is using. Any solution to this problem involves some changes to the application. However, BIP provides support for managing GUI notifications in many useful ways, including support for dual (or even multiple) GUI displays on different computers when a subject is assisted by a person that oversees the capture operation.

8.2 GUI notification callbacks are sent to an application that has subscribed to them by indicating which types of GUI callbacks related to a given attach session (or to a given BSP) it wants to receive, and has subsequently called a capture function on that attach session (or on any attach session of that BSP).

8.3 Such GUI notification callbacks are called primary in BIP. In addition to primary GUI notification callbacks, BIP supports secondary GUI notification callbacks, which are sent to an application (in any endpoint) that previously subscribed to them providing a GUI event subscription UUID. (No GUI event subscription UUID is provided when subscribing to primary GUI notification callbacks.)

8.4 Secondary GUI notification callbacks have the same forms as primary GUI notification callbacks. Also, the same BioAPI function is used to subscribe to primary and to secondary GUI notification callbacks.

8.5 There are two ways in which a secondary GUI notification callback can be sent to an application (called a GUI handler application):

a) on reception of a primary GUI notification callback, the application makes a call to the BioAPI function **BioAPI_NotifyGUIEvent**; the parameters of this function include all the parameters of a GUI notification callback, plus a GUI event subscription UUID; these functions request the framework to generate a secondary GUI notification callback and send it to the application identified by that given GUI event subscription UUID (more precisely, the parameters of this call identify the endpoint, the BSP, the unit ID, and the GUI notification handler);

b) before receiving a primary GUI notification callback, the application has made a call to the BioAPI function **BioAPI_RedirectGUIEvents** by providing a GUI event subscription UUID; the other parameters of this function allow the application to indicate which kind of GUI event notifications are to be redirected to the secondary GUI handler.

9 Examples of possible system configurations

9.1 Figure 1 illustrates three BIP endpoints. In normal BioAPI operation, any application in a system can interwork with any BSP in the same system. This is still possible when using BIP.

9.2 Using BIP, any application in one system (for example, endpoint A) can interwork with any BSP in another system (for example, endpoints B and C), subject to the remote access policy implemented in the endpoints.

9.3 There is no support in this version of the BIP for an application in endpoint B to interwork with a BSP in endpoint C (or vice versa), unless there is a further direct logical connection between endpoints B and C. That is, relaying is not supported by this version of the BIP.

9.4 Figures 2 to 7 show some possible architectures using level-2 implementations of BIP.

NOTE – Figures 2 to 7 are illustrative, and not exhaustive. Many other combinations are possible within the generic architecture of Figure 1.

9.5 Figures 2 and 3 show a BIP endpoint with an application with a locally loaded storage BSP and a remote BIP endpoint with a BSP supporting a biometric sensor, with comparison supported (Figure 2) locally to the application or (Figure 3) remotely.

9.6 Figures 4 and 5 show a BIP endpoint with an application with a locally attached biometric sensor and a remote BIP endpoint with a BSP that supports storage, with comparison implemented (Figure 4) locally to the application or (Figure 5) remotely.

9.7 Figures 6 and 7 show a BIP endpoint with an application but with no locally loaded BSPs, communicating with remote endpoints providing storage and a biometric sensor, with comparison incorporated in either of the two remote endpoints.

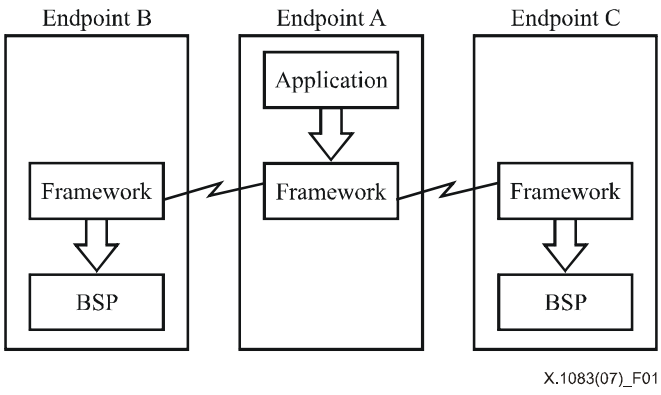

Figure 1 – An application communicating with two other BSPs on two remote systems

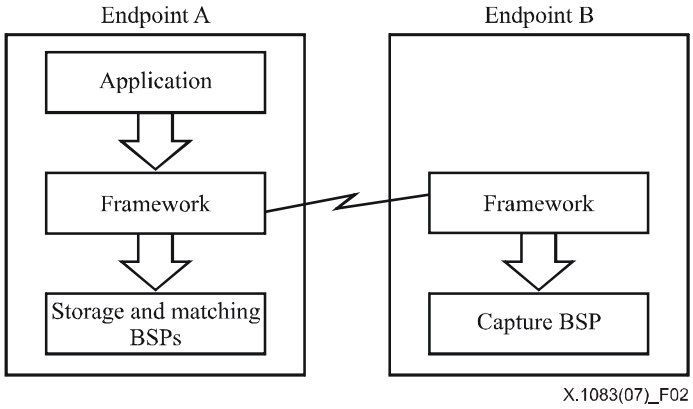

Figure 2 – An application communicating with a remote capture BSP, with all other functions performed locally

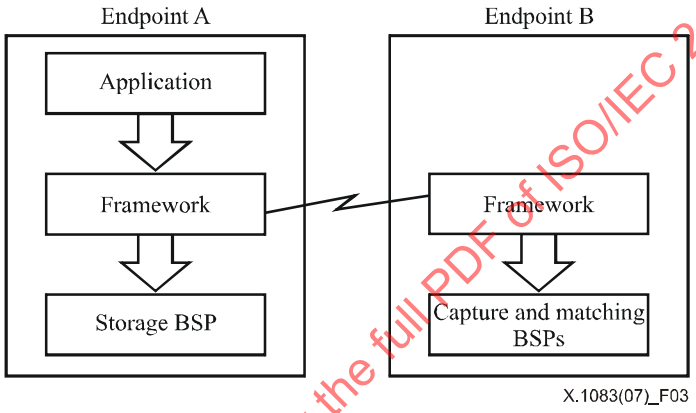

Figure 3 – An application communicating with a remote capture BSP, with remote matching and local storage

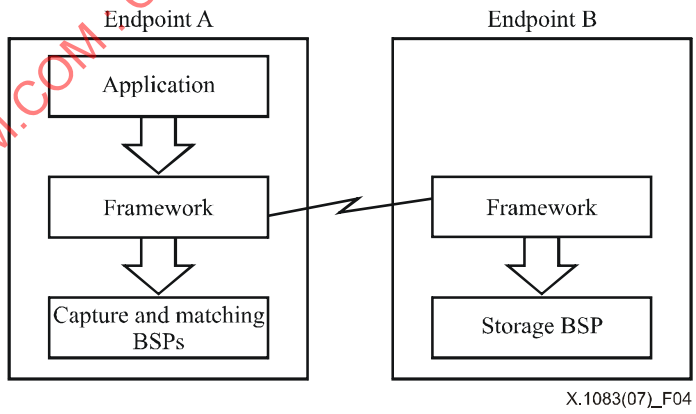

Figure 4 – An application communicating with a remote storage BSP, with all other functions performed locally

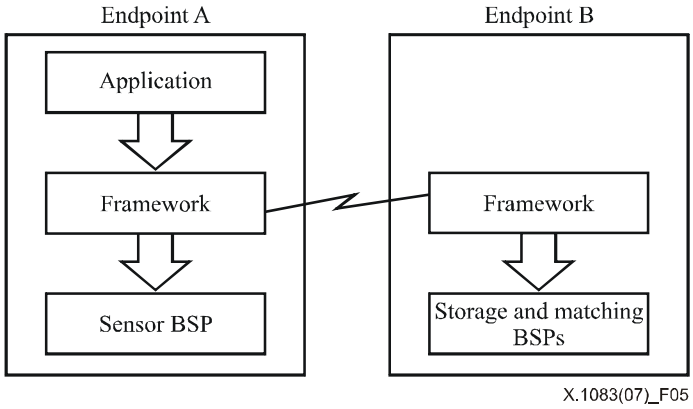

Figure 5 – An application communicating with a remote storage and matching BSP, with a local sensor

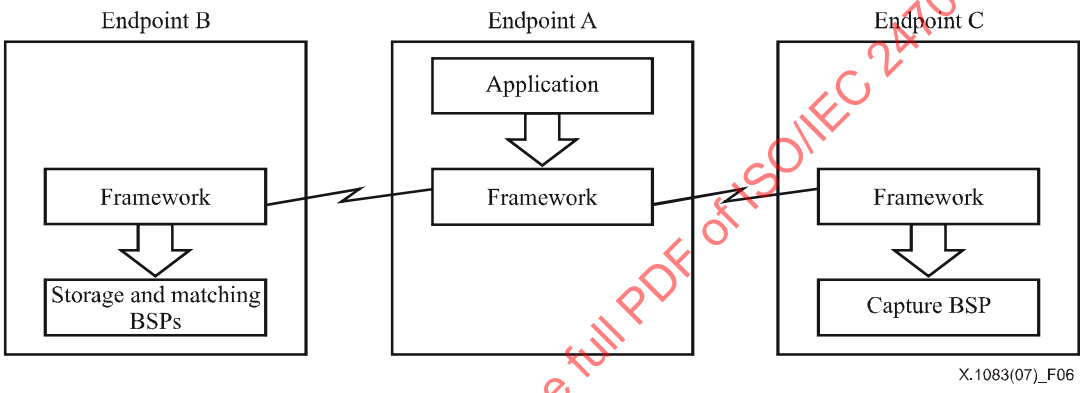

Figure 6 – An application communicating with two remote BSPs, one providing storage and matching and the other providing capture

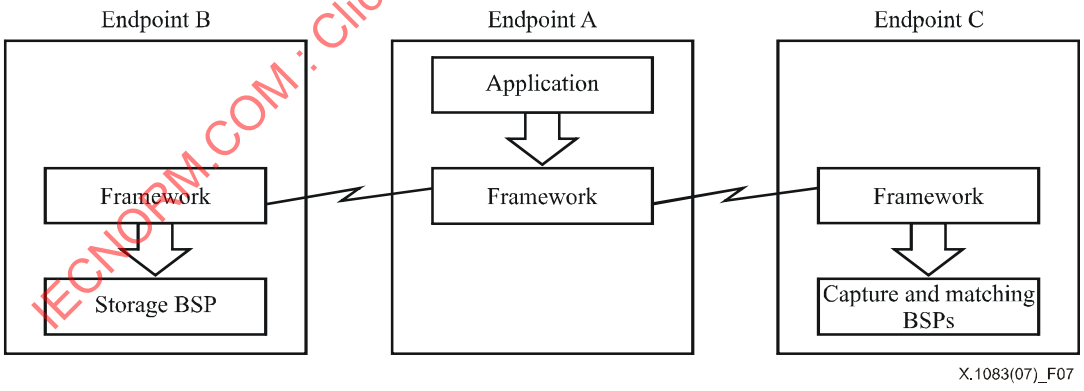

Figure 7 – An application communicating with two remote BSPs, one providing storage and the other providing capture and matching

9.8 BIP can support many other configurations of storage, comparison, and capture BSPs. Essentially, BIP provides transparent (to applications and BSPs) extension of the BioAPI framework across node or process boundaries.

10 BIR format

10.1 A number of BIP message types defined in this Recommendation | International Standard may contain one or more BIR. When transferring such messages, the BIRs can be represented in any CBEFF patron format that meets the following requirements (see 13.16):

a) it supports at least all the data elements and all the abstract values that the BioAPI patron format supports;

b) in case it supports additional data elements, those are all optional; and

c) if it is a complex patron format, it has the ability to carry the data that comprise a simple BIR (and nothing else).

10.2 Each transport protocol binding specifies which patron format is to be used when using that binding. In most cases, that will be either the BioAPI patron format specified in ISO/IEC 19784-1 or the XML patron format specified in ISO/IEC 19785-3.

11 Identification of BIP endpoints, applications, and BSPs

11.1 ISO/IEC 19784-1 clause 10 specifies the use of UUIDs to identify the local BioAPI framework and local BSPs as software products. Those UUIDs are not sufficient to distinguish multiple installations of the same software product on different computers, and obviously they cannot distinguish two instances of the same installed BSP that is available for loading into two different BIP endpoints in the same computer, nor two instances of an installed framework that is common to two different BIP endpoints in the same computer. For these reasons, and also in order to provide a more flexible and Web-friendly naming system for BIP endpoints, this Recommendation | International Standard specifies the use of IRIs to identify BIP endpoints and the use of BSP access UUIDs (as an alternative to BSP product UUIDs) to identify loadable or running BSPs along with their BIP endpoint.

11.2 Since a BIP endpoint contains at most one application, an endpoint IRI is sufficient to identify both a BIP endpoint and the local application in that endpoint (if any). The master endpoint IRI carried in all request BIP messages implicitly identifies the calling application, and the master endpoint IRI carried in all BIP response messages implicitly identifies the application whose function call is being returned.

11.3 In BioAPI, each BSP is identified by a UUID. For the purposes of BIP, this UUID is referred to as the BSP product UUID, to emphasize that this UUID identifies the BSP as a product, rather than as an installed instance or as a running instance of that product. (If the same BSP is installed on different computers, it will have the same BSP product UUID on all of them.) The BSP product UUID is in the BSP schema and is made available to all applications via a call to **BioAPI_EnumBSPs**.

11.4 As a result of BIP endpoint linking, it may happen that the same BSP is available for loading in two or more different endpoints. Although the BSP product UUID uniquely identifies a BSP *within an endpoint* (because the same BSP cannot be registered twice in the same component registry), there is no limit to the number of independent registrations of the same BSP in different component registries (and thus in different endpoints). This implies that the BSP product UUID does not always provide sufficient information as a parameter of **BioAPI_BSPLoad**.

11.5 The function **BioAPI_BSPLoad** expects a UUID as input. In BioAPI, this must be the BSP UUID. BIP extends the semantics of **BioAPI_BSPLoad** by allowing either the BSP product UUID, or a different UUID that identifies both the BSP and the endpoint in which it is registered. The latter is called a *BSP access UUID* and is dynamically generated by the master framework for all (local and remote) BSPs.

11.6 Unlike the BSP product UUID, which may be known to the application from an external source, the BSP access UUID is dynamic and will not become known to the application (in a reliable way) until runtime. The application can do either:

a) call **BioAPI_EnumBSPs** and examine the BSP schemas returned; for each remote BSP (registered and available for loading in the slave endpoint), the BSP schema will contain the BSP product UUID, the endpoint IRI, and the BSP access UUID (among other data); for each local BSP (registered and available for loading in the master endpoint), the BSP schema will contain the BSP product UUID, an empty endpoint IRI, and the BSP access UUID (among other data); the application can choose a BSP schema entry based on the BSP product UUID and the endpoint IRI, read the BSP access UUID from that BSP schema entry, and provide the BSP access UUID as input to **BioAPI_BSPLoad**, **BioAPI_BSPAttach**, etc.; or

b) assume that the desired BSP is registered in exactly one of the various accessible BIP endpoints (the master endpoint and all the slave endpoints), and provide the (known) BSP product UUID as input to **BioAPI_BSPLoad**.

11.7 Of course, method b) will only work if the assumption is correct, but is useful in that it maximizes the chances of successfully running a non-BIP-aware application in a BIP setup. Even in the presence of multiple accessible endpoints, this method can work well, provided that no BSP (with a given BSP product UUID) is available in more than one endpoint. If that is not the case, **BioAPI_BSPLoad** will return an error indicating that the BSP product UUID supplied is ambiguous and the call should be retried by providing the BSP access UUID instead.

12 Overview of BIP exchanges

12.1 Security and privacy provisions

No provision is made in BIP itself for security privacy of exchanges. If security or privacy is required, it can be achieved by using a transport protocol binding that provides security techniques.

12.2 Application invocation of functions on a remote BSP

The BioAPI API functions that produce request BIP messages, the clauses of ISO/IEC 19784-1 specifying the BioAPI functions, the corresponding request BIP message, and the clauses of this Recommendation | International Standard specifying the BIP message are listed in Table 3.

Table 3 – Correspondence between BioAPI functions and BIP message types

BioAPI function	BioAPI clause	BIP message types	Reference
BioAPI_Init	8.1.1	*None*	16.1
BioAPI_InitEndpoint		*None*	16.2
BioAPI_Terminate	8.1.2	masterDeletionEvent notification	16.3
BioAPI_LinkToEndpoint		addMaster request and response	16.4
BioAPI_UnlinkFromEndpoint		deleteMaster request and response	16.5
BioAPI_EnumFrameworks		*None*	16.6
BioAPI_EnumBSPs	8.1.4	*None*	16.7
BioAPI_EnumBFPs	8.1.10	*None*	16.8
BioAPI_BSPLoad	8.1.5	bspLoad request and response	16.9
BioAPI_BSPUnload	8.1.6	bspUnload request and response	16.10
BioAPI_QueryUnits	8.1.9	queryUnits request and response	16.11
BioAPI_QueryBFPs	8.1.11	queryBFPs request and response	16.12
BioAPI_BSPAttach	8.1.7	bspAttach request and response	16.13
BioAPI_BSPDetach	8.1.8	bspDetach request and response	16.14
BioAPI_EnableEvents	8.3.1	enableUnitEvents request and response	16.15
BioAPI_EnableEventNotifications		enableEventNotifications request and response	16.16
BioAPI_ControlUnit	8.1.12	controlUnit request and response	16.17
BioAPI_Control		control request and response	16.18
BioAPI_FreeBIRHandle	8.2.1	freeBIRHandle request and response	16.19
BioAPI_GetBIRFromHandle	8.2.2	getBIRFromHandle request and response	16.20
BioAPI_GetHeaderFromHandle	8.2.3	getHeaderFromHandle request and response	16.21
BioAPI_SubscribeToGUIEvents		subscribeToGUIEvents request and response	16.22
BioAPI_UnsubscribeFromGUIEvents		unsubscribeFromGUIEvents request and response	16.23
BioAPI_QueryGUIEventSubscriptions		queryGUIEventSubscriptions request and response	16.24
BioAPI_NotifyGUISelectEvent		notifyGUISelectEvent request and response	16.25
BioAPI_NotifyGUIStateEvent		notifyGUIStateEvent request and response	16.26
BioAPI_NotifyGUIProgressEvent		notifyGUIProgressEvent request and response	16.27
BioAPI_RedirectGUIEvents		redirectGUIEvents request and response	16.28
BioAPI_UnredirectGUIEvents		unredirectGUIEvents request and response	16.29

Table 3 – Correspondence between BioAPI functions and BIP message types

BioAPI function	BioAPI clause	BIP message types	Reference
BioAPI_Capture	8.4.1	capture request and response	16.30
BioAPI_CreateTemplate	8.4.2	createTemplate request and response	16.31
BioAPI_Process	8.4.3	process request and response	16.32
BioAPI_ProcessWithAuxBIR	8.4.4	processWithAuxBIR request and response	16.33
BioAPI_VerifyMatch	8.4.5	verifyMatch request and response	16.34
BioAPI_IdentifyMatch	8.4.6	identifyMatch request and response	16.35
BioAPI_Enroll	8.4.7	enroll request and response	16.36
BioAPI_Verify	8.4.8	verify request and response	16.37
BioAPI_Identify	8.4.9	identify request and response	16.38
BioAPI_Import	8.4.10	import request and response	16.39
BioAPI_PresetIdentifyPopulation	8.4.11	presetIdentifyPopulation request and response	16.40
BioAPI_Transform		transform request and response	16.41
BioAPI_DbOpen	8.5.1	dbOpen request and response	16.42
BioAPI_DbClose	8.5.2	dbClose request and response	16.43
BioAPI_DbCreate	8.5.3	dbCreate request and response	16.44
BioAPI_DbDelete	8.5.4	dbDelete request and response	16.45
BioAPI_DbSetMarker	8.5.5	dbSetMarker request and response	16.46
BioAPI_DbFreeMarker	8.5.6	dbFreeMarker request and response	16.47
BioAPI_DbStoreBIR	8.5.7	dbStoreBIR request and response	16.48
BioAPI_DbGetBIR	8.5.8	dbGetBIR request and response	16.49
BioAPI_DbGetNextBIR	8.5.9	dbGetNextBIR request and response	16.50
BioAPI_DbDeleteBIR	8.5.10	dbDeleteBIR request and response	16.51
BioAPI_CalibrateSensor	8.6.4	calibrateSensor request and response	16.52
BioAPI_SetPowerMode	8.6.1	setPowerMode request and response	16.53
BioAPI_SetIndicatorStatus	8.6.2	setIndicatorStatus request and response	16.54
BioAPI_GetIndicatorStatus	8.6.3	getIndicatorStatus request and response	16.55
BioAPI_Cancel	8.7.1	cancel request and response	16.57
BioAPI_Free	8.7.2	*none*	16.58
BioAPI_RegisterBSP		registerBSP request, registerBSP response, and bspRegistrationEvent notification	16.59
BioAPI_UnregisterBSP		unregisterBSP request, unregisterBSP response, and bspUnregistrationEvent notification	16.60
BioAPI_RegisterBFP		registerBFP request, registerBFP response, and bfpRegistrationEvent notification	16.61
BioAPI_UnregisterBFP		unregisterBFP request, unregisterBFP response, and bfpUnregistrationEvent notification	16.62
BioAPI_EVENT_HANDLER	7.28	unitEvent notification	17.1
BioAPI_GUI_SELECT_EVENT_HANDLER		guiSelectEvent notification and acknowledgement	17.2
BioAPI_GUI_STATE_EVENT_HANDLER		guiStateEvent notification and acknowledgement	17.3
BioAPI_GUI_PROGRESS_EVENT_HANDLER		guiProgressEvent notification and acknowledgement	17.4

12.3 Application invocation of functions with no associated BIP message

A small number of BioAPI API functions invoked by an application do not identify a BSP and are purely local. They never produce corresponding BIP exchanges, being resolved by information available to the local framework. These are listed in Table 4.

Table 4 – BioAPI functions with no corresponding BIP message type

BioAPI function	BioAPI clause	Source of information	Reference
BioAPI_EnumFrameworks		VisibleEndpoints table	18.2
BioAPI_EnumBSPs	8.1.4	VisibleBSPRegistrations table	18.3
BioAPI_EnumBFPs	8.1.10	VisibleBFPRegistrations table	18.4
BioAPI_Free	8.7.2	ApplicationOwnedMemoryBlocks table	18.13

12.4 Event notifications

Once an application has enabled unit event notification (see BioAPI 9.2.1 & 10.2.1.1/2) from a particular remote BSP (using **BioAPI_EnableEventNotifications**), events that occur in the BSP are transferred to the remote system containing the application by the unitEvent notification BIP message (see 17.1).

13 General provisions

13.1 This clause contains provisions that only apply as invoked by other clauses of this Recommendation | International Standard that reference its individual subclauses.

13.2 In order to create and send a request BIP message of a specified type to a slave endpoint with a specified endpoint IRI, a framework shall create a temporary abstract value of type **BIPMessage** (see clause 14), where:

a) the alternative **request** shall be selected;

b) the component **masterEndpointIRI** shall be set to the local endpoint IRI;

c) the component **slaveEndpointIRI** shall be set to the specified slave endpoint IRI;

d) the component **linkNumber** shall be set to the link number (see 16.4.4 a)) associated with the BIP link from the local endpoint to the specified slave endpoint;

e) the component **requestId** shall be set to the current request identifier associated with the BIP link (see 16.4.4 a)), which shall then be incremented by one (if it is less than 4294967295) or reset to zero (otherwise);

f) the alternative of the component **params** corresponding to the specified request BIP message type shall be selected; and

g) the selected alternative of the component **params** shall be set to the parameter value specified in the invocation of this subclause;

and send (see 13.8) the **BIPMessage** abstract value to the specified BIP slave endpoint.

13.3 In order to create and send a response BIP message corresponding to a given request BIP message, a framework shall create a temporary abstract value of type **BIPMessage** (see clause 14), where:

a) the alternative **response** shall be selected;

b) the components **masterEndpointIRI**, **slaveEndpointIRI**, **linkNumber**, and **requestId** shall be set from the components of the request BIP message with the same names;

c) the alternative of the component **params** with the same name as the alternative of the component **params** that is present in the request BIP message shall be selected;

d) the selected alternative of the component **params** shall be set to the parameter value specified in the invocation of this subclause; and

e) the component **returnValue** shall be set to the return value specified in the invocation of this subclause;

and send (see 13.8) the **BIPMessage** abstract value to the BIP endpoint identified by the value of the component **masterEndpointIRI**.

13.4 In order to create and send a notification BIP message of a specified type to a master endpoint with a specified endpoint IRI, a framework shall create a temporary abstract value of type **BIPMessage** (see clause 14), where:

a) the alternative **notification** shall be selected;

b) the component **slaveEndpointIRI** shall be set to the local endpoint IRI;

c) the component **masterEndpointIRI** shall be set to the specified master endpoint IRI;

d) the component **linkNumber** shall be set to the link number (see 16.4.4 a)) associated with the BIP link from the specified master endpoint to the local endpoint;

e) the component **notificationId** shall be set to the current notification identifier associated with the BIP link (see 16.4.4 a)), which shall then be incremented by one (if it is less than 4294967295) or reset to zero (otherwise);

f) the alternative of the component **params** corresponding to the specified notification BIP message type shall be selected; and

g) the selected alternative of the component **params** shall be set to the parameter value specified in the invocation of this subclause;

and send (see 13.8) the **BIPMessage** abstract value to the BIP endpoint identified by the value of the component **masterEndpointIRI**.

13.5 In order to create and send an acknowledgement BIP message corresponding to a given notification BIP message, a framework shall create a temporary abstract value of type **BIPMessage** (see clause 14), where:

a) the alternative **acknowledgement** shall be selected; and

b) the components **masterEndpointIRI**, **slaveEndpointIRI**, **linkNumber**, and **notificationId** shall be set from the components of the notification BIP message with the same names;

c) the alternative of the component **params** with the same name as the alternative of the component **params** that is present in the notification BIP message shall be selected;

d) the selected alternative of the component **params** shall be set to the parameter value specified in the invocation of this subclause; and

e) the component **returnValue** shall be set to the return value specified in the invocation of this subclause;

and send (see 13.8) the **BIPMessage** abstract value to the BIP endpoint identified by the value of the component **slaveEndpointIRI**.

13.6 Reception of a response BIP message corresponding to a given request BIP message means that a framework receives (see 13.9) a temporary abstract value of type **BIPMessage** (see clause 14), where:

a) the alternative **response** is present;

b) the components **masterEndpointIRI**, **slaveEndpointIRI**, **linkNumber**, and **requestId** have the same values as the components of the request BIP message with the same names; and

c) the alternative of the component **params** that is present has the same name as the alternative of the component **params** that is present in the request BIP message.

13.7 Reception of an acknowledgement BIP message corresponding to a given notification BIP message means that a framework receives (see 13.9) a temporary abstract value of type **BIPMessage** (see clause 14), where:

a) the alternative **acknowledgement** is present;

b) the components **masterEndpointIRI**, **slaveEndpointIRI**, **linkNumber**, and **notificationId** have the same values as the components of the notification BIP message with the same names; and

c) the alternative of the component **params** that is present has the same name as the alternative of the component **params** that is present in the notification BIP message.

13.8 The term "send", as used in this Recommendation | International Standard, usually means the conceptual transfer of a message from a BIP endpoint (sender) to the BIP link between the sender endpoint and another BIP endpoint (receiver). The message being transferred is an abstract value of type **BIPMessage** (see clause 14), which is placed on the BIP link as a result of the send. No assumptions are made with regard to the nature of the BIP link (except that it logically connects two endpoints and is oriented), to the transport mechanism or protocol being used, to the encoding of the BIP message being transferred, and to time-related or space-related aspects of the sending (such as timeouts and queuing).

13.9 The term "receive", as used in this Recommendation | International Standard, usually means the conceptual transfer of a message from the BIP link between a BIP endpoint (sender) and another BIP endpoint (receiver) to the receiver endpoint. The message being transferred is an abstract value of type **BIPMessage** (see clause 14), which is removed from the BIP link and copied to a newly created temporary abstract value (see 13.11) within the receiver endpoint as a result of the receive. No assumptions are made with regard to the nature of the BIP link (except that it logically connects two endpoints and is oriented), to the encoding of the BIP message being transferred, and to time-related or space-related aspects of the reception (such as waiting and queuing).

NOTE – The above means that whenever a sentence such as "receive a corresponding response BIP message" (a response containing the request identifier of the request) occurs normatively in the text, an implementation of a framework may have to wait for an unspecified time before actually obtaining the BIP message. The binding specifications in Annexes A and C provide details of BIP message reception for some particular transports.

13.10 An "internal BioAPI function call" is a conceptual function call made by the framework to a function of its own BioAPI interface in such a way that the call is processed as specified in ISO/IEC 19784-1, and not according to the provisions of this Recommendation | International Standard for a call to the same BioAPI function made by the local application. An internal function call is not required to be an actual C function call (as there is no requirement for a second distinct BioAPI interface inside the framework), but shall be supported by any mechanism which meets the following requirements:

a) before executing an internal call, the framework shall be able to set the parameters of the internal call;

b) while executing the internal call, the framework shall perform all the actions specified in ISO/IEC 19784-1 for an incoming call to the BioAPI function, up to and including the determination of the return value;

c) the actions performed shall not include any of the modifications and additions specified in this Recommendation | International Standard for an incoming call to the same BioAPI function (such as determination of the hosting endpoint); and

d) when the execution of the internal call is completed, the framework shall be able to read the parameter values and the return value of the internal call.

13.11 Many subclauses of this Recommendation | International Standard specify that a temporary abstract value of some ASN.1 type is to be created. All such temporary abstract values are referenced in subsequent subclauses that specify their use. However, there are no subclauses that specify the deletion of a temporary abstract value. A framework may delete a temporary abstract value at any time with the following restrictions:

a) a temporary abstract value created while processing an incoming BioAPI function call from the local application shall not be deleted before the framework has returned control to the local application;

b) a temporary abstract value created while processing an incoming request BIP message from a master endpoint shall not be deleted before the framework has sent a corresponding response BIP message to the master endpoint;

c) a temporary abstract value created while processing an incoming callback from a BSP shall not be deleted before the framework has returned control to the BSP; and

d) a temporary abstract value created while processing an incoming notification BIP message from a slave endpoint shall not be deleted before the framework has either sent a corresponding acknowledgement BIP message to the slave endpoint or has concluded that no acknowledgement BIP message is to be sent.

13.12 Many subclauses of this Recommendation | International Standard specify that, when converting from ASN.1 to C, a variable of some C type (or an array of octets, or an array of elements of some C type) is to be allocated and its address is to be assigned to another C variable. However, there are no subclauses that specify the deletion of such an allocated variable or array. A framework may delete an allocated variable or array at any time with the following restrictions:

a) an allocated variable or array created while processing an incoming BioAPI function call from the local application shall never be deleted except on a subsequent incoming call to **BioAPI_Free** from the local application;

NOTE 1 – Those allocated variables and arrays can only be output parameters of BioAPI functions created by a master framework and being returned to the local application.

b) an allocated variable or array created while processing an incoming request BIP message from a master endpoint shall not be deleted (see 13.14) before the framework has sent a corresponding response BIP message to the master endpoint;

NOTE 2 – Those allocated variables and arrays can only be output parameters of BioAPI functions created by a slave framework and being returned by an internal call.

c) an allocated variable or array created while processing an incoming callback from a BSP shall not be deleted before the framework has returned control to the BSP; and

NOTE 3 – Those allocated variables and arrays can only be input parameters of callback functions created by a master framework and being provided to the local application.

d) an allocated variable or array created while processing an incoming notification BIP message from a slave endpoint shall not be deleted before the framework has either sent a corresponding acknowledgement BIP message to the slave endpoint or has concluded that no acknowledgement BIP message is to be sent.

NOTE 4 – Those allocated variables and arrays can only be input parameters of callback functions created by a master framework and being provided to the local application.

13.13 When an allocated variable or array is created while processing an incoming BioAPI function call from the local application (see 13.12), the framework shall add an entry containing the address of the allocated variable or array to the **ApplicationOwnedMemoryBlocks** table (see 18.13).

13.14 Deletion of an allocated variable or array created while processing an incoming request BIP message (see 13.12 (b)) shall be done by making an internal BioAPI function call (see 13.10) to the function **BioAPI_Free**, where the parameter of the function call shall be the address of the allocated variable or array.

13.15 The version number associated with this edition of this Recommendation | International Standard is (major 1, minor 0).

13.16 When a BIP message containing a BIR is carried within a transport-level message, the BIR may be in any CBEFF patron format that meets the following requirements:

a) it supports at least all the data elements and all the abstract values that the BioAPI patron format supports;

b) in case it supports additional data elements, those are all optional; and

c) if it is a complex patron format, it has the ability to carry the data that comprise a simple BIR (and nothing else);

and shall not be in any patron format that does not meet these requirements.

13.17 When a BIP message to be encoded (at the transport-protocol binding level) contains a **BioAPI-BIR-HEADER** or **BioAPI-BIR** abstract value, then the component **formattedBIR** of that abstract value (which is a BIR in the patron format specified in ISO/IEC 19784-1, Annex B) may be converted, as an option of the transport-protocol binding implementation, to a different patron format prior to encoding. Such conversion is permitted only if the new patron format meets the requirements in 13.16.

13.18 When an incoming BIP message (at the transport-protocol binding level) is found to contain, after decoding, an abstract value of type **BioAPI-BIR-HEADER** or **BioAPI-BIR** whose components **patronFormatOwner** and **patronFormatType** have values different from 257 and 8 (respectively), then the component **formattedBIR** of that abstract value shall either be converted to the patron format specified in ISO/IEC 19784-1, Annex B (only if the original patron format meets the requirements in 13.16 and the implementation knows how to perform the conversion) or shall be left unchanged. If the conversion is performed, the components **patronFormatOwner** and **patronFormatType** shall be set to 257 and 8 (respectively).

14 BIP message syntax

14.1 A BIP message shall consist of an abstract value of the ASN.1 type **BIPMessage** specified below.

```
BIPMessage ::= SEQUENCE {
        nature          CHOICE {
        request             BIPRequest,
        response            BIPResponse,
        notification        BIPNotification,
        acknowledgement     BIPAcknowledgement
        },
    ...
    }
```

```
BIPRequest ::= SEQUENCE {
        slaveEndpointIRI                EndpointIRI,
        masterEndpointIRI               EndpointIRI,
        linkNumber                      UnsignedInt,
        requestId                       UnsignedInt,
        params                          CHOICE {
            addMaster                       AddMaster-RequestParams,
            deleteMaster                    DeleteMaster-RequestParams,
            bspLoad                         BSPLoad-RequestParams,
            bspUnload                       BSPUnload-RequestParams,
            queryUnits                      QueryUnits-RequestParams,
            queryBFPs                       QueryBFPs-RequestParams,
            bspAttach                       BSPAttach-RequestParams,
            bspDetach                       BSPDetach-RequestParams,
            enableUnitEvents                EnableUnitEvents-RequestParams,
            enableEventNotifications        EnableEventNotifications-RequestParams,
            controlUnit                     ControlUnit-RequestParams,
            control                         Control-RequestParams,
            freeBIRHandle                   FreeBIRHandle-RequestParams,
            getBIRFromHandle                GetBIRFromHandle-RequestParams,
            getHeaderFromHandle             GetHeaderFromHandle-RequestParams,
            subscribeToGUIEvents            SubscribeToGUIEvents-RequestParams,
            unsubscribeFromGUIEvents        UnsubscribeFromGUIEvents-RequestParams,
            redirectGUIEvents               RedirectGUIEvents-RequestParams,
            unredirectGUIEvents             UnredirectGUIEvents-RequestParams,
            queryGUIEventSubscriptions      QueryGUIEventSubscriptions-RequestParams,
            notifyGUISelectEvent            NotifyGUISelectEvent-RequestParams,
            notifyGUIStateEvent             NotifyGUIStateEvent-RequestParams,
            notifyGUIProgressEvent          NotifyGUIProgressEvent-RequestParams,
            capture                         Capture-RequestParams,
            createTemplate                  CreateTemplate-RequestParams,
            process                         Process-RequestParams,
            processWithAuxBIR               ProcessWithAuxBIR-RequestParams,
            verifyMatch                     VerifyMatch-RequestParams,
            identifyMatch                   IdentifyMatch-RequestParams,
            enroll                          Enroll-RequestParams,
            verify                          Verify-RequestParams,
            identify                        Identify-RequestParams,
            import                          Import-RequestParams,
            presetIdentifyPopulation        PresetIdentifyPopulation-RequestParams,
            transform                       Transform-RequestParams,
            dbOpen                          DbOpen-RequestParams,
            dbClose                         DbClose-RequestParams,
            dbCreate                        DbCreate-RequestParams,
            dbDelete                        DbDelete-RequestParams,
            dbSetMarker                     DbSetMarker-RequestParams,
            dbFreeMarker                    DbFreeMarker-RequestParams,
            dbStore                         DbStoreBIR-RequestParams,
            dbGetBIR                        DbGetBIR-RequestParams,
            dbGetNextBIR                    DbGetNextBIR-RequestParams,
            dbDeleteBIR                     DbDeleteBIR-RequestParams,
            calibrateSensor                 CalibrateSensor-RequestParams,
            setPowerMode                    SetPowerMode-RequestParams,
            setIndicatorStatus              SetIndicatorStatus-RequestParams,
            getIndicatorStatus              GetIndicatorStatus-RequestParams,
            cancel                          Cancel-RequestParams,
            registerBSP                     RegisterBSP-RequestParams,
            unregisterBSP                   UnregisterBSP-RequestParams,
            registerBFP                     RegisterBFP-RequestParams,
            unregisterBFP                   UnregisterBFP-RequestParams,
            ...
        }
    }
```

```
BIPResponse ::= SEQUENCE {
        slaveEndpointIRI                EndpointIRI,
        masterEndpointIRI               EndpointIRI,
        linkNumber                      UnsignedInt,
        requestId                       UnsignedInt,
        params                          CHOICE {
            addMaster                   AddMaster-ResponseParams,
            deleteMaster                DeleteMaster-ResponseParams,
            bspLoad                     BSPLoad-ResponseParams,
            bspUnload                   BSPUnload-ResponseParams,
            queryUnits                  QueryUnits-ResponseParams,
            queryBFPs                   QueryBFPs-ResponseParams,
            bspAttach                   BSPAttach-ResponseParams,
            bspDetach                   BSPDetach-ResponseParams,
            enableUnitEvents            EnableUnitEvents-ResponseParams,
            enableEventNotifications
                                        EnableEventNotifications-ResponseParams,
            controlUnit                 ControlUnit-ResponseParams,
            control                     Control-ResponseParams,
            freeBIRHandle               FreeBIRHandle-ResponseParams,
            getBIRFromHandle            GetBIRFromHandle-ResponseParams,
            getHeaderFromHandle         GetHeaderFromHandle-ResponseParams,
            subscribeToGUIEvents        SubscribeToGUIEvents-ResponseParams,
            unsubscribeFromGUIEvents    UnsubscribeFromGUIEvents-ResponseParams,
            redirectGUIEvents           RedirectGUIEvents-ResponseParams,
            unredirectGUIEvents         UnredirectGUIEvents-ResponseParams,
            queryGUIEventSubscriptions
                                        QueryGUIEventSubscriptions-ResponseParams,
            notifyGUISelectEvent        NotifyGUISelectEvent-ResponseParams,
            notifyGUIStateEvent         NotifyGUIStateEvent-ResponseParams,
            notifyGUIProgressEvent      NotifyGUIProgressEvent-ResponseParams,
            capture                     Capture-ResponseParams,
            createTemplate              CreateTemplate-ResponseParams,
            process                     Process-ResponseParams,
            processWithAuxBIR           ProcessWithAuxBIR-ResponseParams,
            verifyMatch                 VerifyMatch-ResponseParams,
            identifyMatch               IdentifyMatch-ResponseParams,
            enroll                      Enroll-ResponseParams,
            verify                      Verify-ResponseParams,
            identify                    Identify-ResponseParams,
            import                      Import-ResponseParams,
            presetIdentifyPopulation    PresetIdentifyPopulation-ResponseParams,
            transform                   Transform-ResponseParams,
            dbOpen                      DbOpen-ResponseParams,
            dbClose                     DbClose-ResponseParams,
            dbCreate                    DbCreate-ResponseParams,
            dbDelete                    DbDelete-ResponseParams,
            dbSetMarker                 DbSetMarker-ResponseParams,
            dbFreeMarker                DbFreeMarker-ResponseParams,
            dbStore                     DbStoreBIR-ResponseParams,
            dbGetBIR                    DbGetBIR-ResponseParams,
            dbGetNextBIR                DbGetNextBIR-ResponseParams,
            dbDeleteBIR                 DbDeleteBIR-ResponseParams,
            calibrateSensor             CalibrateSensor-ResponseParams,
            setPowerMode                SetPowerMode-ResponseParams,
            setIndicatorStatus          SetIndicatorStatus-ResponseParams,
            getIndicatorStatus          GetIndicatorStatus-ResponseParams,
            cancel                      Cancel-ResponseParams,
            registedBSP                 RegisterBSP-ResponseParams,
            unregisterBSP               UnregisterBSP-ResponseParams,
            registerBFP                 RegisterBFP-ResponseParams,
            unregisterBFP               UnregisterBFP-ResponseParams,
            ...
        },
        returnValue                             BioAPI-RETURN
    }
```

```
BIPNotification ::= SEQUENCE {
        masterEndpointIRI               EndpointIRI,
        slaveEndpointIRI                EndpointIRI,
        linkNumber                      UnsignedInt,
        notificationId                  UnsignedInt,
        params                          CHOICE {
            masterDeletionEvent         MasterDeletionEvent-NotificationParams,
            unitEvent                   UnitEvent-NotificationParams,
            guiSelectEvent              GUISelectEvent-NotificationParams,
            guiStateEvent               GUIStateEvent-NotificationParams,
            guiProgressEvent            GUIProgressEvent-NotificationParams,
            bspRegistrationEvent        BSPRegistrationEvent-NotificationParams,
            bspUnregistrationEvent      BSPUnregistrationEvent-NotificationParams,
            bfpRegistrationEvent        BFPRegistrationEvent-NotificationParams,
            bfpUnregistrationEvent      BFPUnregistrationEvent-NotificationParams,
            ...
        }
    }

BIPAcknowledgement ::= SEQUENCE {
        masterEndpointIRI               EndpointIRI,
        slaveEndpointIRI                EndpointIRI,
        linkNumber                      UnsignedInt,
        notificationId                  UnsignedInt,
        params                          CHOICE {
            guiSelectEvent              GUISelectEvent-AcknowledgementParams,
            guiStateEvent               GUIStateEvent-AcknowledgementParams,
            guiProgressEvent            GUIProgressEvent-AcknowledgementParams,
            ...
        },
        returnValue                         BioAPI-RETURN
    }
```

14.2 This and all other ASN.1 type definitions are assumed to be in an environment of automatic tags. The formal specification of the ASN.1 module with these types is in Annex F.

15 BioAPI and BIP types

15.1 Integers

15.1.1 The following integer ASN.1 types are defined in BIP:

```
UnsignedByte        ::= INTEGER (0..max-unsigned-byte)
UnsignedShort       ::= INTEGER (0..max-unsigned-short)
UnsignedInt         ::= INTEGER (0..max-unsigned-int)
SignedInt           ::= INTEGER (min-signed-int..max-signed-int)
MemoryAddress       ::= INTEGER
```

where the value references are defined as follows:

```
max-unsigned-byte   INTEGER ::= 255
max-unsigned-short  INTEGER ::= 65535
max-unsigned-int    INTEGER ::= 4294967295
min-signed-int      INTEGER ::= -2147483648
max-signed-int      INTEGER ::= 2147483647
```

15.1.2 The built-in ASN.1 type **INTEGER** occurs, always with a range constraint, in correspondence to an integer C type (most frequently **uint8_t)**. The set of values of the constrained ASN.1 type is either the same or a subset of the set of values of the C type (there may be one or more unconvertible C values). Conversion between the ASN.1 type and the C type (in both directions) shall be done by mapping an integer ASN.1 abstract value to the corresponding (signed or unsigned) C integer value. Clause 33 applies when an unconvertible integer C value is encountered.

15.1.3 The ASN.1 type **UnsignedByte** occurs in correspondence to the C type **uint8_t**. The set of values is the same in both languages. Conversion between the ASN.1 type and the C type (in both directions) shall be done by mapping an integer ASN.1 abstract value to the corresponding 8-bit C unsigned integer value.

15.1.4 The ASN.1 type **UnsignedShort** occurs in correspondence to the C type **uint16_t**. The set of values is the same in both languages. Conversion between the ASN.1 type and the C type (in both directions) shall be done by mapping an integer ASN.1 abstract value to the corresponding 16-bit C unsigned integer value.

15.1.5 The ASN.1 type **UnsignedInt** occurs in correspondence to the C type **uint32_t**. The set of values is the same in both languages. Conversion between the ASN.1 type and the C type (in both directions) shall be done by mapping an integer ASN.1 abstract value to the corresponding 32-bit C unsigned integer value.

15.1.6 The ASN.1 type **SignedInt** occurs in correspondence to the C type **int32_t**. The set of values is the same in both languages. Conversion between the ASN.1 type and the C type (in both directions) shall be done by mapping an integer ASN.1 abstract value to the corresponding 32-bit C signed integer value.

15.1.7 The ASN.1 type **MemoryAddress** occurs in correspondence to the C type **void*** or a C function pointer type. The mapping between ASN.1 values of type **MemoryAddress** and C pointer values is not specified in this Recommendation | International Standard.

NOTE – Any implementation-defined mapping between **MemoryAddress** values and C pointer values is acceptable so long as each pointer value is mapped to a different **MemoryAddress** integer and that integer is mapped back to the same pointer value. The ASN.1 type **MemoryAddress** does not occur in the content of any BIP messages exchanged between BIP endpoints, and therefore its abstract values are never encoded.

15.2 Character strings

15.2.1 The built-in ASN.1 type **UTF8String** occurs in correspondence to the C type **uint8_t*** where the C variable points to a zero-terminated octet array containing a UTF-8 encoded character string.

15.2.2 Conversion from the C pointer variable to the ASN.1 component shall be done as follows:

a) if the C pointer variable has the value **NULL** and the ASN.1 component is **OPTIONAL**, then the ASN.1 component shall be absent;

b) if the C pointer variable has the value **NULL** and the ASN.1 component is not **OPTIONAL**, then the C value is unconvertible and clause 33 applies;

c) if the C pointer variable has a value different from **NULL**, then the content of the octet array pointed to by the C variable up to the first zero-valued octet (exclusive) shall be interpreted as the UTF-8 encoding of a character string, and the ASN.1 component shall be set to that character string.

15.2.3 Conversion from the ASN.1 component to the C pointer variable shall be done as follows:

a) if the ASN.1 component is **OPTIONAL** and is absent, then the C pointer variable shall be set to **NULL**;

b) if the ASN.1 component is present, then, calling *L* the length (in octets) of the UTF-8 encoding of the ASN.1 abstract character string, a newly allocated array of *L* + 1 octets shall be filled with that UTF-8 encoding followed by a zero-valued octet, and the C variable shall be set to the address of that octet array.

15.3 Uniform resource identifiers designating BIP endpoints

15.3.1 The following ASN.1 type is defined in BIP:

```
EndpointIRI ::= VisibleString (CONSTRAINED BY
                    {--The string shall conform to the "absolute-IRI" grammar--
                     --defined in IETF RFC 3987--)
```

15.3.2 The ASN.1 type **EndpointIRI** occurs in correspondence to the C type **uint8_t*** where the C variable points to a zero-terminated octet array containing a uniform resource identifier.

15.3.3 Conversion from the C pointer variable to the ASN.1 component shall be done as follows:

a) if the C pointer variable has the value **NULL** then the ASN.1 component shall be set to the local endpoint IRI;

b) otherwise, the content of the octet array pointed to by the C variable up to the first zero-valued octet (exclusive) shall be interpreted as the ASCII codes of a character string, and the ASN.1 component shall be set to that character string.

15.3.4 Conversion from the ASN.1 component to the C pointer variable shall be done as follows:

a) if the ASN.1 component contains the local endpoint IRI, then the C pointer variable shall be set to **NULL**;

b) otherwise, calling *L* the length of the ASN.1 abstract character string, a newly allocated array of *L* + 1 octets shall be filled with the ASCII codes of the ASN.1 character string followed by a zero-valued octet, and the C variable shall be set to the address of that octet array.

15.4 Type BioAPI_BFP_LIST_ELEMENT

15.4.1 This C type is defined in BioAPI as follows:

```
typedef struct _bioapi_bfp_list_element {
        BioAPI_CATEGORY BFPCategory;
        BioAPI_UUID BFPUuid;
    } BioAPI_BFP_LIST_ELEMENT;
```

15.4.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BFP-LIST-ELEMENT ::= SEQUENCE {

        category                BioAPI-CATEGORY,
        bfpProductUuid          BioAPI-UUID
    }
```

15.4.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 5.

Table 5 – Mapping between members of the C type BioAPI_BFP_LIST_ELEMENT and components of the ASN.1 type BioAPI-BFP-LIST-ELEMENT

Member of the C type	Component of the ASN.1 type	References
BFPCategory	Category	15.21
BFPUuid	bfpProductUuid	15.58

NOTE – No "**HostingEndpointIRI**" member is present in the C type **BioAPI_BFP_LIST_ELEMENT**. This C type is used in the **BioAPI_QueryBFPs** function, and all BFPs in the BFP list returned by that function are in the same hosting BIP endpoint, which is the BIP endpoint where the BSP is running.

15.5 Type BioAPI_BFP_SCHEMA

15.5.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_bfp_schema {
        BioAPI_UUID BFPProductUuid;
        BioAPI_CATEGORY BFPCategory;
        BioAPI_STRING BFPDescription;
        uint8_t *Path;
        BioAPI_VERSION SpecVersion;
        BioAPI_STRING ProductVersion;
        BioAPI_STRING Vendor;
        BioAPI_BIR_BIOMETRIC_DATA_FORMAT *BFPSupportedFormats;
        uint32_t NumSupportedFormats;
        BioAPI_BIR_BIOMETRIC_TYPE FactorsMask;
        BioAPI_UUID BFPPropertyUuid;
        BioAPI_DATA BFPProperty;
        uint8_t *HostingEndpointIRI;
    } BioAPI_BFP_SCHEMA;
```

15.5.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BFP-SCHEMA ::= SEQUENCE {
        bfpProductUuid          BioAPI-UUID,
        category                BioAPI-CATEGORY,
        description             BioAPI-STRING,
        path                    UTF8String,
        specVersion             BioAPI-VERSION,
        productVersion          BioAPI-STRING,
        vendor                  BioAPI-STRING,
        supportedFormats        SEQUENCE (SIZE(0..max-unsigned-int)) OF
                                    format BioAPI-BIR-BIOMETRIC-DATA-FORMAT,
        factorsMask             BioAPI-BIR-BIOMETRIC-TYPE,
        propertyUuid            BioAPI-UUID,
        property                BioAPI-DATA,
        hostingEndpointIRI      EndpointIRI
}
```

15.5.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 6.

Table 6 – Mapping between members of the C type BioAPI_BFP_SCHEMA and components of the ASN.1 type BioAPI-BFP-SCHEMA

Member of the C type	Component of the ASN.1 type	References
BFPProductUuid	bfpProductUuid	15.58
BFPCategory	Category	15.21
BFPDescription	Description	15.53
Path	Path	15.2
SpecVersion	specVersion	15.59
ProductVersion	productVersion	15.53
Vendor	Vendor	15.53
BFPSupportedFormats, NumSupportedFormats	supportedFormats	15.5.4 and 15.5.5
FactorsMask	factorsMask	15.10
BFPPropertyUuid	propertyUuid	15.58
BFPProperty	Property	15.22
HostingEndpointIRI	hostingEndpointIRI	15.3

15.5.4 Conversion from the pair of C members **BFPSupportedFormats**/**NumSupportedFormats** to the ASN.1 component **supportedFormats** shall be done as follows: Calling *N* the value of the member **NumSupportedFormats**, each of the first *N* elements (of type **BioAPI_BIR_BIOMETRIC_DATA_FORMAT** – see 15.8) in the array pointed to by the member **BFPSupportedFormats** shall be converted, in order, to an element of the component **supportedFormats** as specified in 15.8. The component **supportedFormats** shall have exactly *N* elements.

15.5.5 Conversion from the ASN.1 component **supportedFormats** to the pair of C members **BFPSupportedFormats**/**NumSupportedFormats** shall be done as follows: Calling *N* the number of elements of the component **supportedFormats**, a newly allocated array of *N* elements of type **BioAPI_BIR_BIOMETRIC_DATA_FORMAT** (see 15.8) shall be filled by converting each element of the component **supportedFormats**, in order, to an element of the array as specified in 15.8. The member **BFPSupportedFormats** shall be set to the address of the array, and the member **NumSupportedFormats** shall be set to *N*.

NOTE – While the C type **BioAPI_BSP_SCHEMA** (see 15.19) includes a BSP access UUID, there is no BFP access UUID in the C type **BioAPI_BFP_SCHEMA**. Since BFPs are not accessed directly by any BioAPI function, a BFP access UUID is not needed. The endpoint IRI is included in this C type because an application may wish to call **BioAPIEnumBFPs** to obtain the list of BFPs, learn about a BFP available in some BIP endpoint, and then remotely load a BSP in that BIP endpoint in an attempt to use that BFP.

15.6 Type BioAPI_BIR

15.6.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_bir {
        BioAPI_BIR_HEADER Header;
        BioAPI_DATA BiometricData;
        BioAPI_DATA SecurityBlock;
} BioAPI_BIR;
```

15.6.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR ::= SEQUENCE {
        patronFormatOwner  UnsignedShort,
        patronFormatType        UnsignedShort,
        formattedBIR            OCTET STRING
}
```

15.6.3 Conversion from the C type to the ASN.1 type shall be done by serializing the value of the C type to a BioAPI BIR (see ISO/IEC 19784-1, Annex B) placing the resulting BIR in the **formattedBIR** octet string and setting **patronFormatOwner** and **patronFormatType** to 257 and 8 (respectively).

NOTE – In any abstract value of this ASN.1 type, the **formattedBIR** component consists of the octets of a BIR in the format specified in Annex B of BioAPI. A transport protocol binding implementation can either include the octet string in the encoding as it is, or convert it to another patron format prior to encoding (see 13.17).

15.6.4 Conversion from the ASN.1 type to the C type shall be done by interpreting the content of the **formattedBIR** octet string as a BioAPI BIR (see ISO/IEC 19784-1, Annex B) and deserializing it into a value of the C type, provided that **patronFormatOwner** and **patronFormatType** have the values 257 and 8 (respectively). If these components have different values, then the framework shall create and send a response BIP message that matches the request BIP message, with the return value set to **BioAPIERR_PATRON_FORMAT_NOT_SUPPORTED**.

NOTE – This occurs when the implementation of the transport protocol binding has received a message containing a BIR in an unsupported or disallowed patron format, which cannot be converted to the BioAPI BIR patron format (see 13.18).

15.7 Type BioAPI_BIR_ARRAY_POPULATION

15.7.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_bir_array_population {
        uint32_t NumberOfMembers;
        BioAPI_BIR *Members;
} BioAPI_BIR_ARRAY_POPULATION;
```

15.7.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR-ARRAY-POPULATION ::= SEQUENCE {
        members     SEQUENCE (SIZE(0..max-unsigned-int)) OF
                                member BioAPI-BIR
}
```

15.7.3 Conversion from the pair of C members **NumberOfMembers**/**Members** to the ASN.1 component **members** shall be done as follows: Calling *N* the value of the member **NumberOfMembers**, each of the first *N* elements (of type **BioAPI_BIR** – see 15.6) in the array pointed to by the member **Members** shall be converted, in order, to an element of the component **members** as specified in 15.6. The component **members** shall have exactly *N* elements.

15.7.4 Conversion from the ASN.1 component **members** to the pair of C members **NumberOfMembers**/ **Members** shall be done as follows: Calling *N* the number of elements of the component **members**, a newly allocated array of *N* elements of type **BioAPI_BIR** (see 15.6) shall be filled by converting each element of the component **members**, in order, to an element of the array as specified in 15.6. The member **Members** shall be set to the address of the array, and the member **NumberOfMembers** shall be set to *N*.

15.8 Type BioAPI_BIR_BIOMETRIC_DATA_FORMAT

15.8.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_bir_biometric_data_format {
        uint16_t FormatOwner;
        uint16_t FormatType;
} BioAPI_BIR_BIOMETRIC_DATA_FORMAT;
```

15.8.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR-BIOMETRIC-DATA-FORMAT ::= SEQUENCE {
        formatOwner         UnsignedShort,
        formatType          UnsignedShort
}
```

15.8.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 7.

Table 7 – Mapping between members of the C type BioAPI_BIR_BIOMETRIC_DATA_FORMAT and components of the ASN.1 type BioAPI-BIR-BIOMETRIC-DATA-FORMAT

Member of the C type	Component of the ASN.1 type	References
FormatOwner	formatOwner	15.1.4
FormatType	formatType	15.1.4

15.9 Type BioAPI_BIR_BIOMETRIC_PRODUCT_ID

15.9.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_bir_biometric_product_ID {
        uint16_t ProductOwner;
        uint16_t ProductType;
} BioAPI_BIR_BIOMETRIC_PRODUCT_ID;
```

15.9.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR-BIOMETRIC-PRODUCT-ID ::= SEQUENCE {
        productOwner        UnsignedShort,
        productType         UnsignedShort
}
```

15.9.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 8.

Table 8 – Mapping between members of the C type BioAPI_BIR_BIOMETRIC_PRODUCT_ID and components of the ASN.1 type BioAPI-BIR-BIOMETRIC-PRODUCT-ID

Member of the C type	Component of the ASN.1 type	References
ProductOwner	productOwner	15.1.4
ProductType	productType	15.1.4

15.10 Type BioAPI_BIR_BIOMETRIC_TYPE

15.10.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_BIR_BIOMETRIC_TYPE;
```

15.10.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_NO_BIOTYPE_AVAILABLE              (0x00000000)
#define BioAPI_TYPE_MULTIPLE_BIOMETRIC_TYPES     (0x00000001)
#define BioAPI_TYPE_FACE                         (0x00000002)
#define BioAPI_TYPE_VOICE                        (0x00000004)
#define BioAPI_TYPE_FINGER                       (0x00000008)
#define BioAPI_TYPE_IRIS                         (0x00000010)
#define BioAPI_TYPE_RETINA                       (0x00000020)
#define BioAPI_TYPE_HAND_GEOMETRY                (0x00000040)
#define BioAPI_TYPE_SIGNATURE_SIGN               (0x00000080)
#define BioAPI_TYPE_KEYSTROKE                    (0x00000100)
#define BioAPI_TYPE_LIP_MOVEMENT                 (0x00000200)
#define BioAPI_TYPE_GAIT                         (0x00001000)
#define BioAPI_TYPE_VEIN                         (0x00002000)
#define BioAPI_TYPE_DNA                          (0x00004000)
#define BioAPI_TYPE_EAR                          (0x00008000)
#define BioAPI_TYPE_FOOT                         (0x00010000)
#define BioAPI_TYPE_SCENT                        (0x00020000)
#define BioAPI_TYPE_OTHER                        (0x40000000)
#define BioAPI_TYPE_PASSWORD                     (0x80000000)
```

15.10.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR-BIOMETRIC-TYPE ::= BIT STRING {

                typeMultipleBiometricTypes      (0),
                typeFace                        (1),
                typeVoice                       (2),
                typeFinger                      (3),
                typeIris                        (4),
                typeRetina                      (5),
                typeHandGeometry                (6),
                typeSignatureSign               (7),
                typeKeystroke                   (8),
                typeLipMovement                 (9),

                typeGait                        (12),
                typeVein                        (13),
                typeDNA                         (14),
                typeEar                         (15),
                typeFoot                        (16),
                typeScent                       (17),
                typeOther                       (30),
                typePassword                    (31)
        } (SIZE(32))
```

15.10.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as follows: Each bit of the ASN.1 bit string shall be mapped to a bit of the C unsigned integer. The leading bit of the bit string (bit 0) shall be mapped to the least significant bit of the unsigned integer (corresponding to the value 0x00000001), and the remaining thirty-one bits shall be mapped in order.

15.11 Type BioAPI_BIR_DATA_TYPE

15.11.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_BIR_DATA_TYPE;
```

15.11.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_BIR_DATA_TYPE_RAW                 (0x01)
#define BioAPI_BIR_DATA_TYPE_INTERMEDIATE        (0x02)
#define BioAPI_BIR_DATA_TYPE_PROCESSED           (0x04)
#define BioAPI_BIR_DATA_TYPE_ENCRYPTED           (0x10)
#define BioAPI_BIR_DATA_TYPE_SIGNED              (0x20)
#define BioAPI_BIR_INDEX_PRESENT                 (0x80)
```

15.11.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR-DATA-TYPE ::= SEQUENCE {
        processedLevel      ENUMERATED {
                raw,
                intermediate,
                processed,
                ...
        },
        flags               BIT STRING {
                encrypted           (0),
                signed              (1),
                index-present       (3)
        } (SIZE                     (4))
}
```

15.11.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as follows:

a) the component flags of the ASN.1 type shall be mapped to the four most significant bits of the C unsigned integer; the leading bit (bit 0) of this component shall be mapped to the least significant of the four most significant bits of the C unsigned integer (the one corresponding to the value 0x10), and the remaining three bits shall be mapped in order; and

b) the component **processedLevel** shall be mapped to the four least significant bits of the C unsigned integer in accordance with Table 9.

Table 9 – Mapping between the least significant four bits of the C type BioAPI_BIR_DATA_TYPE and the component processedLevel of the ASN.1 type BioAPI-BIR-DATA-TYPE

Value at bit position 8 0x01	Value at bit position 7 0x02	Value at bit position 6 0x04	Value at bit position 5 0x08	Value of the ASN.1 component processedLevel
1	0	0	0	raw
0	1	0	0	intermediate
0	0	1	0	processed
other combinations of values				*none – the C value is unconvertible and clause 33 applies*

15.12 Type BioAPI_BIR_HANDLE

15.12.1 This C type is defined in BioAPI as follows:

```
typedef int32_t BioAPI_BIR_HANDLE;
```

15.12.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define  BioAPI_INVALID_BIR_HANDLE          (-1)
#define  BioAPI_UNSUPPORTED_BIR_HANDLE      (-2)
```

15.12.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR-HANDLE ::= SignedInt
```

15.12.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as specified in 15.1.6.

15.13 Type BioAPI_BIR_HEADER

15.13.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_bir_header {
        BioAPI_VERSION HeaderVersion;
        BioAPI_BIR_DATA_TYPE Type;
        BioAPI_BIR_BIOMETRIC_DATA_FORMAT Format;
        BioAPI_QUALITY Quality;
        BioAPI_BIR_PURPOSE Purpose;
        BioAPI_BIR_BIOMETRIC_TYPE FactorsMask;
        BioAPI_BIR_BIOMETRIC_PRODUCT_ID ProductID;
```

```
        BioAPI_DTG CreationDTG;
        BioAPI_BIR_SUBTYPE Subtype;
        BioAPI_DATE ExpirationDate;
        BioAPI_BIR_SECURITY_BLOCK_FORMAT SBFormat;
        BioAPI_UUID Index;
    } BioAPI_BIR_HEADER;
```

15.13.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR-HEADER ::= SEQUENCE {
        patronFormatOwner       UnsignedShort,
        patronFormatType        UnsignedShort,
        formattedBIR            OCTET STRING
}
```

15.13.3 Conversion from the C type to the ASN.1 type shall be done by serializing the value of the C type to a BioAPI BIR as specified in ISO/IEC 19784-1, Annex B, placing the resulting BIR in the **formattedBIR** component, and setting **patronFormatOwner** and **patronFormatType** to 257 and 8 (respectively).

NOTE – In any abstract value of this ASN.1 type, the **formattedBIR** component consists of the octets of a BIR in the format specified in Annex B of BioAPI, with an empty BDB and SB. A transport protocol binding implementation can either include the octet string in the encoding as it is, or convert it to another patron format prior to encoding (see 13.17).

15.13.4 Conversion from the ASN.1 type to the C type shall be done by interpreting the content of the **formattedBIR** component as a BioAPI BIR as specified in ISO/IEC 19784-1, Annex B, and deserializing it into a value of the C type, provided that **patronFormatOwner** and **patronFormatType** have the values 257 and 8 (respectively). If these components have different values, then the framework shall create and send a response BIP message that matches the request BIP message, with the return value set to **BioAPIERR_PATRON_FORMAT_NOT_SUPPORTED**.

NOTE – This occurs when the implementation of the transport protocol binding has received a message containing a BIR in an unsupported or disallowed patron format, which cannot be converted to the BioAPI BIR patron format (see 13.18).

15.14 Type BioAPI_BIR_PURPOSE

15.14.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_BIR_PURPOSE;
```

15.14.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define  BioAPI_PURPOSE_VERIFY                                  (1)
#define  BioAPI_PURPOSE_IDENTIFY                                (2)
#define  BioAPI_PURPOSE_ENROLL                                  (3)
#define  BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY            (4)
#define  BioAPI_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY          (5)
#define  BioAPI_PURPOSE_AUDIT                                   (6)
#define  BioAPI_PURPOSE_ANY                                     (7)
```

15.14.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR-PURPOSE ::= ENUMERATED {
        verify,
        identify,
        enroll,
        enrollVerify,
        enrollIdentify,
        audit,
        any,
        ...
}
```

15.14.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done in accordance with Table 10.

Table 10 – Mapping between individual values of the C type BioAPI_BIR_PURPOSE and abstract values of the ASN.1 type BioAPI-BIR-PURPOSE

Value of the C type	Abstract value of the ASN.1 type
1 (BioAPI_PURPOSE_VERIFY)	verify
2 (BioAPI_PURPOSE_IDENTIFY)	identify
3 (BioAPI_PURPOSE_ENROLL)	enroll
4 (BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY)	enrollVerify
5 (BioAPI_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY)	enrollIdentify
6 (BioAPI_PURPOSE_AUDIT)	audit
7 (BioAPI_PURPOSE_ANY)	any
other values	*none – the C value is unconvertible and clause 33 applies*

15.15 Type BioAPI_BIR_SECURITY_BLOCK_FORMAT

15.15.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_bir_security_block_format {
        uint16_t SecurityFormatOwner;
        uint16_t SecurityFormatType;
} BioAPI_BIR_SECURITY_BLOCK_FORMAT;
```

15.15.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR-SECURITY-BLOCK-FORMAT ::= SEQUENCE {
        formatOwner         UnsignedShort,
        formatType          UnsignedShort
}
```

15.15.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 11.

Table 11 – Mapping between members of the C type BioAPI_BIR_SECURITY_BLOCK_FORMAT and components of the ASN.1 type BioAPI-BIR-SECURITY-BLOCK-FORMAT

Member of the C type	Component of the ASN.1 type	References
SecurityFormatOwner	formatOwner	15.1.4
SecurityFormatType	formatType	15.1.4

15.16 Type BioAPI_BIR_SUBTYPE

15.16.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_BIR_SUBTYPE;
```

15.16.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define  BioAPI_BIR_SUBTYPE_VEIN_ONLY_MASK      (0x80)
#define  BioAPI_BIR_SUBTYPE_LEFT_MASK           (0x01)
#define  BioAPI_BIR_SUBTYPE_RIGHT_MASK          (0x02)

#define  BioAPI_BIR_SUBTYPE_THUMB               (0x04)
#define  BioAPI_BIR_SUBTYPE_POINTERFINGER       (0x08)
#define  BioAPI_BIR_SUBTYPE_MIDDLEFINGER        (0x10)
#define  BioAPI_BIR_SUBTYPE_RINGFINGER          (0x20)
#define  BioAPI_BIR_SUBTYPE_LITTLEFINGER        (0x40)

#define  BioAPI_BIR_SUBTYPE_VEIN_PALM           (0x04)
#define  BioAPI_BIR_SUBTYPE_VEIN_BACKOFHAND     (0x08)
#define  BioAPI_BIR_SUBTYPE_VEIN_WRIST          (0x10)

#define BioAPI_NO_SUBTYPE_AVAILABLE             (0x00)
```

15.16.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR-SUBTYPE ::= CHOICE {
        any-subtype    BIT STRING {
                left                    (0),
                right                   (1),
                thumb                   (2),
                pointerFinger           (3),
                middleFinger            (4),
                ringFinger              (5),
                littleFinger            (6)} (SIZE(7)),
        vein-only-subtype BIT STRING {
                left                    (0),
                right                   (1),
                veinPalm                (2),
                veinBackofhand          (3),
                veinWrist               (4)} (SIZE(7))

        }
```

15.16.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as follows: Each bit of the ASN.1 bit string shall be mapped to a bit of the C unsigned integer. The leading bit of the bit string (bit 0) shall be mapped to the least significant bit of the unsigned integer (corresponding to the value 0x01), and the remaining seven bits shall be mapped in order.

15.17 Type BioAPI_BIR_SUBTYPE_MASK

15.17.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_BIR_SUBTYPE_MASK;
```

15.17.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_BIR_SUBTYPE_LEFT_BIT                     (0x00000001)
#define BioAPI_BIR_SUBTYPE_RIGHT_BIT                    (0x00000002)
#define BioAPI_BIR_SUBTYPE_LEFT_THUMB_BIT               (0x00000004)
#define BioAPI_BIR_SUBTYPE_LEFT_POINTERFINGER_BIT       (0x00000008)
#define BioAPI_BIR_SUBTYPE_LEFT_MIDDLEFINGER_BIT        (0x00000010)
#define BioAPI_BIR_SUBTYPE_LEFT_RINGFINGER_BIT          (0x00000020)
#define BioAPI_BIR_SUBTYPE_LEFT_LITTLEFINGER_BIT        (0x00000040)
#define BioAPI_BIR_SUBTYPE_RIGHT_THUMB_BIT              (0x00000080)
#define BioAPI_BIR_SUBTYPE_RIGHT_POINTERFINGER_BIT      (0x00000100)
#define BioAPI_BIR_SUBTYPE_RIGHT_MIDDLEFINGER_BIT       (0x00000200)
#define BioAPI_BIR_SUBTYPE_RIGHT_RINGFINGER_BIT         (0x00000400)
#define BioAPI_BIR_SUBTYPE_RIGHT_LITTLEFINGER_BIT       (0x00000800)
#define BioAPI_BIR_SUBTYPE_LEFT_VEIN_PALM_BIT           (0x00001000)
#define BioAPI_BIR_SUBTYPE_LEFT_VEIN_BACKOFHAND_BIT     (0x00002000)
#define BioAPI_BIR_SUBTYPE_LEFT_VEIN_WRIST_BIT          (0x00004000)
#define BioAPI_BIR_SUBTYPE_RIGHT_VEIN_PALM_BIT          (0x00008000)
#define BioAPI_BIR_SUBTYPE_RIGHT_VEIN_BACKOFHAND_BIT    (0x00010000)
#define BioAPI_BIR_SUBTYPE_RIGHT_VEIN_WRIST_BIT         (0x00020000)
```

15.17.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BIR-SUBTYPE-MASK ::= BIT STRING {
                left                    (0),
                right                   (1),
                left-thumb              (2),
                left-pointerfinger      (3),
                left-middlefinger       (4),
                left-ringfinger         (5),
                left-littlefinger       (6),
                right-thumb             (7),
                right-pointerfinger     (8),
                right-middlefinger      (9),
                right-ringfinger        (10),
                right-littlefinger      (11),
                left-vein-palm          (12),
                left-vein-backofhand    (13),
```

```
        left-vein-wrist          (14),
        right-vein-palm          (15),
        right-vein-backofhand    (16),
        right-vein-wrist         (17)
    } (SIZE(32))
```

15.17.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as follows: Each bit of the ASN.1 bit string shall be mapped to a bit of the C unsigned integer. The leading bit of the bit string (bit 0) shall be mapped to the least significant bit of the unsigned integer (corresponding to the value 0x00000001), and the remaining thirty-one bits shall be mapped in order.

15.18 Type BioAPI_BOOL

15.18.1 This C type is defined in BioAPI as follows:

```
typedef int8_t BioAPI_BOOL;
```

15.18.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_FALSE        (0)
#define BioAPI_TRUE         (!BioAPI_FALSE)
```

15.18.3 The corresponding ASN.1 type in BIP is the built-in type **BOOLEAN**.

15.18.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done in accordance with Table 12.

Table 12 – Mapping between individual values of the C type BioAPI_BOOL and abstract values of the ASN.1 type BOOLEAN

Value of the C type	Abstract value of the ASN.1 type
0 (BioAPI_FALSE)	FALSE
1 (BioAPI_TRUE)	TRUE
other values	*none – the C value is unconvertible and clause 33 applies*

15.19 Type BioAPI_BSP_SCHEMA

15.19.1 This C type is defined in BioAPI as follows:

```
typedef struct _bioapi_bsp_schema {
        BioAPI_UUID BSPProductUuid;
        BioAPI_STRING BSPDescription;
        uint8_t *Path;
        BioAPI_VERSION SpecVersion;
        BioAPI_STRING ProductVersion;
        BioAPI_STRING Vendor;
        BioAPI_BIR_BIOMETRIC_DATA_FORMAT *BSPSupportedFormats;
        uint32_t NumSupportedFormats;
        BioAPI_BIR_BIOMETRIC_TYPE FactorsMask;
        BioAPI_OPERATIONS_MASK Operations;
        BioAPI_OPTIONS_MASK Options;
        BioAPI_FMR PayloadPolicy;
        uint32_t MaxPayloadSize;
        int32_t DefaultVerifyTimeout;
        int32_t DefaultIdentifyTimeout;
        int32_t DefaultCaptureTimeout;
        int32_t DefaultEnrollTimeout;
        int32_t DefaultCalibrateTimeout;
        uint32_t MaxBSPDbSize;
        uint32_t MaxIdentify;
        uint8_t *HostingEndpointIRI;
        BioAPI_UUID BSPAccessUuid;
} BioAPI_BSP_SCHEMA;
```

15.19.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-BSP-SCHEMA ::= SEQUENCE {
        bspProductUuid          BioAPI-UUID,
        description             BioAPI-STRING,
        path                    UTF8String,
        specVersion             BioAPI-VERSION,
        productVersion          BioAPI-STRING,
        vendor                  BioAPI-STRING,
        supportedFormats        SEQUENCE (SIZE(0..max-unsigned-int)) OF
                                    format BioAPI-BIR-BIOMETRIC-DATA-FORMAT,
        factorsMask             BioAPI-BIR-BIOMETRIC-TYPE,
        operations              BioAPI-OPERATIONS-MASK,
        options                 BioAPI-OPTIONS-MASK,
        payloadPolicy           BioAPI-FMR,
        maxPayloadSize          UnsignedInt,
        defaultVerifyTimeout    SignedInt,
        defaultIdentifyTimeout  SignedInt,
        defaultCaptureTimeout   SignedInt,
        defaultEnrollTimeout    SignedInt,
        defaultCalibrateTimeout SignedInt,
        maxBSPDbSize            UnsignedInt,
        maxIdentify             UnsignedInt,
        hostingEndpointIRI      EndpointIRI,
        bspAccessUuid           BioAPI-UUID
}
```

15.19.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 13.

Table 13 – Mapping between members of the C type BioAPI_BSP_SCHEMA and components of the ASN.1 type BioAPI-BSP-SCHEMA

Member of the C type	Component of the ASN.1 type	References
BSPProductUuid	bspProductUuid	15.58
BSPDescription	description	15.53
Path	Path	15.2
SpecVersion	specVersion	15.59
ProductVersion	productVersion	15.53
Vendor	vendor	15.53
BSPSupportedFormats, NumSupportedFormats	supportedFormats	15.19.4 and 15.19.5
FactorsMask	factorsMask	15.10
Operations	operations	15.48
Options	options	15.49
PayloadPolicy	payloadPolicy	15.32
MaxPayloadSize	maxPayloadSize	15.1.5
DefaultVerifyTimeout	defaultVerifyTimeout	15.1.6
DefaultIdentifyTimeout	defaultIdentifyTimeout	15.1.6
DefaultCaptureTimeout	defaultCaptureTimeout	15.1.6
DefaultEnrollTimeout	defaultEnrollTimeout	15.1.6
DefaultCalibrateTimeout	defaultCalibrateTimeout	15.1.6
MaxBSPDbSize	maxBSPDbSize	15.1.5
MaxIdentify	maxIdentify	15.1.5
HostingEndpointIRI	hostingEndpointIRI	15.3
BSPAccessUuid	bspAccessUuid	15.58

15.19.4 Conversion from the pair of C members **BSPSupportedFormats**/**NumSupportedFormats** to the ASN.1 component **supportedFormats** shall be done as follows: Calling *N* the value of the member **NumSupportedFormats**, each of the first *N* elements (of type **BioAPI_BIR_BIOMETRIC_DATA_FORMAT** – see 15.8) in the array pointed to by the member **BSPSupportedFormats** shall be converted, in order, to an element of the component **supportedFormats** as specified in 15.8. The component **supportedFormats** shall have exactly *N* elements.

15.19.5 Conversion from the ASN.1 component **supportedFormats** to the pair of C members **BSPSupportedFormats**/**NumSupportedFormats** shall be done as follows: Calling *N* the number of elements of the component **supportedFormats**, a newly allocated array of *N* elements of type **BioAPI_BIR_BIOMETRIC_DATA_FORMAT** (see 15.8) shall be filled by converting each element of the component **supportedFormats**, in order, to an element of the array as specified in 15.8. The member **BSPSupportedFormats** shall be set to the address of the array, and the member **NumSupportedFormats** shall be set to *N*.

15.20 Type BioAPI_CANDIDATE

15.20.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_candidate {
        BioAPI_IDENTIFY_POPULATION_TYPE Type;
        union {
            BioAPI_UUID *BIRInDataBase;
            uint32_t *BIRInArray;
            } BIR;
        BioAPI_FMR  FMRAchieved
    } BioAPI_CANDIDATE;
```

15.20.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-CANDIDATE ::= SEQUENCE {
        bir                             CHOICE {
            birInDatabase               BioAPI-UUID,
            birInArray                  UnsignedInt,
            birInPresetArray            UnsignedInt
        },
        fmrAchieved                     BioAPI-FMR
    }
```

15.20.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 14.

Table 14 – Mapping between members of the C type BioAPI_CANDIDATE and components of the ASN.1 type BioAPI-CANDIDATE

Member of the C type	Component of the ASN.1 type	References
Type, BIR	Bir	15.20.4 and 15.20.5
FMRAchieved	fmrAchieved	15.32

15.20.4 Conversion from the pair of C members **Type**/**BIR** to the ASN.1 component **bir** shall be done by using Table 15 to perform the following actions (in order):

a) determine which alternative of the member **BIR** is present, based on the value of **Type**;

b) select the alternative of the component **bir** corresponding to the value of **Type**;

c) convert the C member to the alternative of the component **bir**.

15.20.5 Conversion from the ASN.1 component **bir** to the pair of C members **Type**/**BIR** shall be done by using Table 15 to perform the following actions (in order):

a) set the member **Type** based on the alternative of the component **bir** that is present;

b) select the corresponding alternative of the member **BIR**;

c) convert the ASN.1 component to the C member.

Table 15 – Mapping between alternatives of the member BIR of the C type BioAPI_CANDIDATE and alternatives of the component bir of the ASN.1 type BioAPI-CANDIDATE

Value of the member Type	Alternative of the member BIR	Alternative of the component bir	References
1 (BioAPI_DB_TYPE)	BIRInDataBase	birInDatabase	clause 19 in conjunction with 15.58
2 (BioAPI_ARRAY_TYPE)	BIRInArray	birInArray	clause 19 in conjunction with 15.1.5
3 (BioAPI_PRESET_ARRAY_TYPE)	BIRInPresetArray	birInPresetArray	clause 19 in conjunction with 15.1.5
other values	*none – the C value is unconvertible and clause 33 applies*		

15.21 Type BioAPI_CATEGORY

15.21.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_CATEGORY;
```

15.21.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_CATEGORY_ARCHIVE              (0x00000001)
#define BioAPI_CATEGORY_COMPARISON_ALG       (0x00000002)
#define BioAPI_CATEGORY_PROCESSING_ALG       (0x00000004)
#define BioAPI_CATEGORY_SENSOR               (0x00000008)
```

15.21.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-CATEGORY ::= ENUMERATED {
                    archive,
                    comparisonAlgorithm,
                    processingAlgorithm,
                    sensor,
                    ...
                }
```

15.21.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done in accordance with Table 16.

Table 16 – Mapping between individual values of the C type BioAPI_CATEGORY and abstract values of the ASN.1 type BioAPI-CATEGORY

Value of the C type	Abstract value of the ASN.1 type
1 (BioAPI_CATEGORY_ARCHIVE)	archive
2 (BioAPI_CATEGORY_COMPARISON_ALG)	comparisonAlgorithm
4 (BioAPI_CATEGORY_PROCESSING_ALG)	processingAlgorithm
8 (BioAPI_CATEGORY_SENSOR)	sensor
other values	*none – the C value is unconvertible and clause 33 applies*

15.22 Type BioAPI_DATA

15.22.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_data{
        uint32_t Length;
        void *Data;
        } BioAPI_DATA;
```

15.22.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-DATA ::= OCTET STRING (SIZE(0..max-unsigned-int))
```

15.22.3 Conversion from the C variable to the ASN.1 component shall be done as follows:

a) if the member **Data** of the C variable has the value **NULL** and the ASN.1 component is **OPTIONAL**, then the ASN.1 component shall be absent;

b) if the member **Data** of the C variable has the value **NULL** and the ASN.1 component is not **OPTIONAL**, then the C value is unconvertible and clause 33 applies;

c) if the member **Data** of the C variable has a value different from **NULL**, then, calling *L* the value of the member **Length** of the C type, the first *L* octets of the octet array pointed to by the member **Data** of the C type shall form an octet string to be assigned to the ASN.1 abstract value.

15.22.4 Conversion from the ASN.1 component to the C variable shall be done as follows:

a) if the ASN.1 component is **OPTIONAL** and is absent, then the member **Data** of the C variable shall be set to **NULL**, and the member **Length** of the C variable shall be set to 0;

b) if the ASN.1 component is present, then, calling *L* the length of the ASN.1 abstract octet string, a newly allocated array of *L* octets shall be filled with the octets of that octet string; the member **Data** of the C variable shall be set to the address of that octet array, and the member **Length** of the C variable shall be set to *L*.

15.23 Type BioAPI_DATE

15.23.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_date {
        uint16_t Year;
        uint8_t Month;
        uint8_t Day;
    } BioAPI_DATE;
```

15.23.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-DATE ::= SEQUENCE {
        year        INTEGER (0 | 1900..9999),
        month       INTEGER (0..12),
        day         INTEGER (0..31)
    }
```

15.23.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 17.

Table 17 – Mapping between members of the C type BioAPI_DATE and components of the ASN.1 type BioAPI-DATE

Member of the C type	Component of the ASN.1 type	References
Year	year	15.1.2
Month	month	15.1.2
Day	day	15.1.2

15.23.4 The abstract value of type **BioAPI-DATE** in which the three components **year**, **month**, and **day** have the value 0 is valid. Apart from that particular abstract value, all **year** values outside the range 1900 to 9999, all **month** values outside the range 1 to 12, and all **day** values outside the range 1 to 31 are unconvertible, and clause 33 applies when such a value is encountered.

15.24 Type BioAPI_DB_ACCESS_TYPE

15.24.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_DB_ACCESS_TYPE;
```

15.24.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_DB_ACCESS_READ       (0x00000001)
#define BioAPI_DB_ACCESS_WRITE      (0x00000002)
```

15.24.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-DB-ACCESS-TYPE ::= BIT STRING {
        read        (0),
        write       (1)
    } (SIZE(32))
```

15.24.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as follows: Each bit of the ASN.1 bit string shall be mapped to a bit of the C unsigned integer. The leading bit of the bit string (bit 0) shall be mapped to the least significant bit of the unsigned integer (corresponding to the value 0x00000001), and the remaining thirty-one bits shall be mapped in order.

15.25 Type BioAPI_DB_MARKER_HANDLE

15.25.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_DB_MARKER_HANDLE;
```

15.25.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-DB-MARKER-HANDLE ::= UnsignedInt
```

15.25.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done as specified in 15.1.5.

15.26 Type BioAPI_DB_HANDLE

15.26.1 This C type is defined in BioAPI as follows:

```
typedef int32_t BioAPI_DB_HANDLE;
```

15.26.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_DB_INVALID_HANDLE      (-1)
#define BioAPI_DB_DEFAULT_HANDLE      (0)
#define BioAPI_DB_DEFAULT_UUID_PTR    (NULL)
```

15.26.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-DB-HANDLE ::= SignedInt
```

15.26.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as specified in 15.1.6.

15.27 Type BioAPI_DBBIR_ID

15.27.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_dbbir_id {
                BioAPI_DB_HANDLE DbHandle;
                BioAPI_UUID KeyValue;
        } BioAPI_DBBIR_ID;
```

15.27.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-DBBIR-ID ::= SEQUENCE {
                dbHandle            BioAPI-DB-HANDLE,
                keyValue            BioAPI-UUID
        }
```

15.27.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 18.

Table 18 – Mapping between members of the C type BioAPI_DBBIR_ID and components of the ASN.1 type BioAPI-DBBIR-ID

Member of the C type	Component of the ASN.1 type	References
DbHandle	dbHandle	15.26
KeyValue	keyValue	15.58

15.28 Type BioAPI_DTG

15.28.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_DTG {
        BioAPI_DATE Date;
        BioAPI_TIME Time;
    } BioAPI_DTG;
```

15.28.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-DTG ::= SEQUENCE {
        date        BioAPI-DATE,
        time        BioAPI-TIME
    }
```

15.28.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 19.

Table 19 – Mapping between members of the C type BioAPI_DTG and components of the ASN.1 type BioAPI-DTG

Member of the C type	Component of the ASN.1 type	References
Date	date	15.23
Time	time	15.54

15.29 Type BioAPI_ERROR_INFO

15.29.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_error_info {
        int32_t ErrorCode;
        BioAPI_STRING ErrorMessage;
        BioAPI_STRING ErrorSourceName;
} BioAPI_ERROR_INFO;
```

15.29.2 There is no corresponding ASN.1 type in BIP.

NOTE – This C type is used only in the function BioAPI_GetLastErrorInfo (see 16.56), which has no related BIP message type.

15.30 Type BioAPI_EVENT

15.30.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_EVENT;
```

15.30.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_NOTIFY_INSERT            (1)
#define BioAPI_NOTIFY_REMOVE            (2)
#define BioAPI_NOTIFY_FAULT             (3)
#define BioAPI_NOTIFY_SOURCE_PRESENT    (4)
#define BioAPI_NOTIFY_SOURCE_REMOVED    (5)
```

15.30.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-UNIT-EVENT-TYPE ::= ENUMERATED {
        insert,
        remove,
        fault,
        sourcePresent,
        sourceRemoved,
        ...
    }
```

15.30.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done in accordance with Table 20.

Table 20 – Mapping between individual values of the C type BioAPI_EVENT and abstract values of the ASN.1 type BioAPI-UNIT-EVENT-TYPE

Value of the C type	Abstract value of the ASN.1 type
1 (BioAPI_NOTIFY_INSERT)	insert
2 (BioAPI_NOTIFY_REMOVE)	remove
3 (BioAPI_NOTIFY_FAULT)	fault
4 (BioAPI_NOTIFY_SOURCE_PRESENT)	sourcePresent
5 (BioAPI_NOTIFY_SOURCE_REMOVED)	sourceRemoved
other values	*none – the C value is unconvertible and clause 33 applies*

15.31 Type BioAPI_EVENT_MASK

15.31.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_EVENT_MASK;
```

15.31.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define  BioAPI_NOTIFY_INSERT_BIT                (0x00000001)
#define  BioAPI_NOTIFY_REMOVE_BIT                (0x00000002)
#define  BioAPI_NOTIFY_FAULT_BIT                 (0x00000004)
#define  BioAPI_NOTIFY_SOURCE_PRESENT_BIT        (0x00000008)
#define  BioAPI_NOTIFY_SOURCE_REMOVED_BIT        (0x00000010)
```

15.31.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-UNIT-EVENT-TYPE-MASK ::= BIT STRING {
                insert                  (0),
                remove                  (1),
                fault                   (2),
                sourcePresent           (3),
                sourceRemoved           (4)
        } (SIZE(32))
```

15.31.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as follows: Each bit of the ASN.1 bit string shall be mapped to a bit of the C unsigned integer. The leading bit of the bit string (bit 0) shall be mapped to the least significant bit of the unsigned integer (corresponding to the value 0x00000001), and the remaining thirty-one bits shall be mapped in order.

15.32 Type BioAPI_FMR

15.32.1 This C type is defined in BioAPI as follows:

```
typedef int32_t BioAPI_FMR;
```

15.32.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-FMR ::= SignedInt
```

15.32.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done as specified in 15.1.6.

15.33 Type BioAPI_FRAMEWORK_SCHEMA

15.33.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_framework_schema {
                BioAPI_UUID FwProductUuid;
                BioAPI_STRING FwDescription;
                uint8_t *Path;
                BioAPI_VERSION SpecVersion;
                BioAPI_STRING ProductVersion;
                BioAPI_STRING Vendor;
                BioAPI_UUID FwPropertyUuid;
                BioAPI_DATA FwProperty;
                uint8_t *HostingEndpointIRI;
        } BioAPI_FRAMEWORK_SCHEMA;
```

15.33.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-FRAMEWORK-SCHEMA ::= SEQUENCE {
        fwProductUuid           BioAPI-UUID,
        description             BioAPI-STRING,
        path                    UTF8String,
        specVersion             BioAPI-VERSION,
        productVersion          BioAPI-STRING,
        vendor                  BioAPI-STRING,
        propertyUuid            BioAPI-UUID,
        property                BioAPI-DATA,
        hostingEndpointIRI      EndpointIRI
    }
```

15.33.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 21.

Table 21 – Mapping between members of the C type BioAPI_FRAMEWORK_SCHEMA and components of the ASN.1 type BioAPI-FRAMEWORK-SCHEMA

Member of the C type	Component of the ASN.1 type	References
FwProductUuid	fwProductUuid	15.58
FwDescription	description	15.53
Path	path	15.2
SpecVersion	specVersion	15.59
ProductVersion	productVersion	15.53
Vendor	vendor	15.53
FwPropertyUuid	propertyUuid	15.58
FwProperty	property	15.22
HostingEndpointIRI	hostingEndpointIRI	15.3

15.34 Type BioAPI_GUI_BITMAP

15.34.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_gui_bitmap {
        BioAPI_BIR_SUBTYPE_MASK SubtypeMask;
        uint32_t Width;
        uint32_t Height;
        BioAPI_DATA Bitmap;
    } BioAPI_GUI_BITMAP;
```

15.34.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-GUI-BITMAP ::= SEQUENCE {
        subtypeMask             BioAPI-BIR-SUBTYPE-MASK,
        width                   UnsignedInt,
        height                  UnsignedInt,
        bitmap                  BioAPI-DATA OPTIONAL
    }
```

15.34.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 22.

Table 22 – Mapping between members of the C type BioAPI_GUI_BITMAP and components of the ASN.1 type BioAPI-GUI-BITMAP

Member of the C type	Component of the ASN.1 type	References
SubtypeMask	subtypemask	15.17
Width	width	15.1.5
Height	height	15.1.5
Bitmap	bitmap	15.22

15.35 Type BioAPI_GUI_BITMAP_ARRAY

15.35.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_gui_bitmap_array {
        uint32_t NumberOfMembers;
        BioAPI_GUI_BITMAP *GuiBitmaps;
} BioAPI_GUI_BITMAP_ARRAY;
```

15.35.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-GUI-BITMAP-ARRAY ::= SEQUENCE {
        guiBitmaps          SEQUENCE (SIZE(0..max-unsigned-int)) OF
                                    guiBitmap BioAPI-GUI-BITMAP
}
```

15.35.3 Conversion from the pair of C members **NumberOfMembers**/**Members** to the ASN.1 component **members** shall be done as follows: Calling *N* the value of the member **NumberOfMembers**, each of the first *N* elements (of type **BioAPI_GUI_BITMAP** – see 15.34) in the array pointed to by the member **Members** shall be converted, in order, to an element of the component **members** as specified in 15.34. The component **members** shall have exactly *N* elements.

15.35.4 Conversion from the ASN.1 component **members** to the pair of C members **NumberOfMembers**/**Members** shall be done as follows: Calling *N* the number of elements of the component **members**, a newly allocated array of *N* elements of type **BioAPI_GUI_BITMAP** (see 15.34) shall be filled by converting each element of the component **members**, in order, to an element of the array as specified in 15.34. The member **Members** shall be set to the address of the array, and the member **NumberOfMembers** shall be set to *N*.

15.36 Type BioAPI_GUI_EVENT_SUBSCRIPTION

15.36.1 This C type is defined in BioAPI as follows:

```
typedef struct _bioapi_gui_event_subscription {
        const uint8_t *SubscriberEndpointIRI;
        BioAPI_UUID GUIEventSubscriptionUuid;
        BioAPI_BOOL GUISelectEventSubscribed;
        BioAPI_BOOL GUIStateEventSubscribed;
        BioAPI_BOOL GUIProgressEventSubscribed;
} BioAPI_GUI_EVENT_SUBSCRIPTION;
```

15.36.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-GUI-EVENT-SUBSCRIPTION ::= SEQUENCE {
        subscriberEndpointIRI           EndpointIRI,
        guiEventSubscriptionUuid        BioAPI-UUID,
        guiSelectEventSubscribed        BOOLEAN,
        guiStateEventSubscribed         BOOLEAN,
        guiProgressEventSubscribed      BOOLEAN
}
```

15.36.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 23.

Table 23 – Mapping between members of the C type BioAPI_GUI_EVENT_SUBSCRIPTION and components of the ASN.1 type BioAPI-GUI-EVENT-SUBSCRIPTION

Member of the C type	Component of the ASN.1 type	References
SubscriberEndpointIRI	subscriberEndpointIRI	15.3
GUIEventSubscriptionUuid	guiEventSubscriptionUuid	15.58
GUISelectEventSubscribed	guiSelectEventSubscribed	15.18
GUIStateEventSubscribed	guiStateEventSubscribed	15.18
GUIProgressEventSubscribed	guiProgressEventSubscribed	15.18

15.37 Type BioAPI_GUI_MOMENT

15.37.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_GUI_MOMENT;
```

15.37.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_GUI_MOMENT_BEFORE_START      (1)
#define BioAPI_GUI_MOMENT_DURING            (2)
#define BioAPI_GUI_MOMENT_AFTER_END         (3)
```

15.37.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-GUI-MOMENT ::= ENUMERATED {
                beforeStart,
                during,
                afterEnd,
                ...
        }
```

15.37.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done in accordance with Table 24.

Table 24 – Mapping between individual values of the C type BioAPI_GUI_MOMENT and abstract values of the ASN.1 type BioAPI-GUI-MOMENT

Value of the C type	Abstract value of the ASN.1 type
1 (BioAPI_GUI_MOMENT_BEFORE_START)	before-start
2 (BioAPI_GUI_MOMENT_DURING)	during
3 (BioAPI_GUI_MOMENT_AFTER_END)	after-end
other values	*none – the C value is unconvertible and clause 33 applies*

15.38 Type BioAPI_GUI_ENROLL_TYPE

15.38.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_GUI_ENROLL_TYPE;
```

15.38.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_GUI_ENROLL_TYPE_TEST_VERIFY          (0x00000001)
#define BioAPI_GUI_ENROLL_TYPE_MULTIPLE_CAPTURE     (0x00000002)
```

15.38.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-GUI-ENROLL-TYPE ::= BIT STRING {
                testVerify          (0),
                multipleCapture     (1)
        } (SIZE(32))
```

15.38.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as follows: Each bit of the ASN.1 bit string shall be mapped to a bit of the C unsigned integer. The leading bit of the bit string (bit 0) shall be mapped to the least significant bit of the unsigned integer (corresponding to the value 0x00000001), and the remaining thirty-one bits shall be mapped in order.

15.39 Type BioAPI_GUI_OPERATION

15.39.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_GUI_OPERATION;
```

15.39.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_GUI_OPERATION_CAPTURE              (1)
#define BioAPI_GUI_OPERATION_PROCESS              (2)
#define BioAPI_GUI_OPERATION_CREATETEMPLATE       (3)
#define BioAPI_GUI_OPERATION_VERIFYMATCH          (4)
#define BioAPI_GUI_OPERATION_IDENTIFYMATCH        (5)
#define BioAPI_GUI_OPERATION_VERIFY               (6)
#define BioAPI_GUI_OPERATION_IDENTIFY             (7)
#define BioAPI_GUI_OPERATION_ENROLL               (8)
```

15.39.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-GUI-OPERATION ::= ENUMERATED {
                capture,
                process,
                createtemplate,
                verifymatch,
                identifymatch,
                verify,
                identify,
                enroll,
                ...
        }
```

15.39.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done in accordance with Table 25.

Table 25 – Mapping between individual values of the C type BioAPI_GUI_OPERATION and abstract values of the ASN.1 type BioAPI-GUI-OPERATION

Value of the C type	Abstract value of the ASN.1 type
1 (BioAPI_GUI_OPERATION_CAPTURE)	capture
2 (BioAPI_GUI_OPERATION_PROCESS)	process
3 (BioAPI_GUI_OPERATION_CREATETEMPLATE)	createtemplate
4 (BioAPI_GUI_OPERATION_VERIFYMATCH)	verifymatch
5 (BioAPI_GUI_OPERATION_IDENTIFYMATCH)	identifymatch
6 (BioAPI_GUI_OPERATION_VERIFY)	verify
7 (BioAPI_GUI_OPERATION_IDENTIFY)	identify
8 (BioAPI_GUI_OPERATION_ENROLL)	enroll
other values	*none – the C value is unconvertible and clause 33 applies*

15.40 Type BioAPI_GUI_RESPONSE

15.40.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_GUI_RESPONSE;
```

15.40.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_GUI_RESPONSE_DEFAULT               (0)
#define BioAPI_GUI_RESPONSE_OP_COMPLETE           (1)
#define BioAPI_GUI_RESPONSE_OP_CANCEL             (2)
#define BioAPI_GUI_RESPONSE_CYCLE_START           (3)
#define BioAPI_GUI_RESPONSE_CYCLE_RESTART         (4)
#define BioAPI_GUI_RESPONSE_SUBOP_START           (5)
#define BioAPI_GUI_RESPONSE_SUBOP_NEXT            (6)
#define BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE     (7)
#define BioAPI_GUI_RESPONSE_PROGRESS_ABORT        (8)
#define BioAPI_GUI_RESPONSE_RECAPTURE             (9)
```

15.40.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-GUI-RESPONSE ::= ENUMERATED {
                default,
                opComplete,
                opCancel,
                cycleStart,
                cycleRestart,
                subopStart,
                subopNext,
                progressContinue,
                progressCancel,
                recapture,
                ...
        }
```

15.40.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done in accordance with Table 26.

Table 26 – Mapping between individual values of the C type BioAPI_GUI_RESPONSE and abstract values of the ASN.1 type BioAPI-GUI-RESPONSE

Value of the C type	Abstract value of the ASN.1 type
0 (BioAPI_GUI_RESPONSE_DEFAULT)	default
1 (BioAPI_GUI_RESPONSE_OP_COMPLETE)	opComplete
2 (BioAPI_GUI_RESPONSE_OP_CANCEL)	opCancel
3 (BioAPI_GUI_RESPONSE_CYCLE_START)	cycleStart
4 (BioAPI_GUI_RESPONSE_CYCLE_RESTART)	cycleRestart
5 (BioAPI_GUI_RESPONSE_SUBOP_START)	subopStart
6 (BioAPI_GUI_RESPONSE_SUBOP_NEXT)	subopNext
7 (BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE)	progressContinue
8 (BioAPI_GUI_RESPONSE_PROGRESS_ABORT)	progressAbort
9 (BioAPI_GUI_RESPONSE_RECAPTURE)	recapture
other values	*none – the C value is unconvertible and clause 33 applies*

15.41 Type BioAPI_GUI_SUBOPERATION

15.41.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_GUI_SUBOPERATION;
```

15.41.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_GUI_SUBOPERATION_CAPTURE             (1)
#define BioAPI_GUI_SUBOPERATION_PROCESS             (2)
#define BioAPI_GUI_SUBOPERATION_CREATETEMPLATE      (3)
#define BioAPI_GUI_SUBOPERATION_VERIFYMATCH         (4)
#define BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH       (5)
```

15.41.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-GUI-SUBOPERATION ::= ENUMERATED {
                capture,
                process,
                createtemplate,
                verifymatch,
                identifymatch,
                ...
        }
```

15.41.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done in accordance with Table 27.

Table 27 – Mapping between individual values of the C type BioAPI_GUI_SUBOPERATION and abstract values of the ASN.1 type BioAPI-GUI-SUBOPERATION

Value of the C type	Abstract value of the ASN.1 type
1 (BioAPI_GUI_SUBOPERATION_CAPTURE)	capture
2 (BioAPI_GUI_SUBOPERATION_PROCESS)	process
3 (BioAPI_GUI_SUBOPERATION_CREATETEMPLATE)	createtemplate
4 (BioAPI_GUI_SUBOPERATION_VERIFYMATCH)	verifymatch
5 (BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH)	identifymatch
Other values	*none – the C value is unconvertible and clause 33 applies*

15.42 Type BioAPI_HANDLE

15.42.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_HANDLE;
```

15.42.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-HANDLE ::= UnsignedInt
```

15.42.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done as specified in 15.1.5.

15.43 Type BioAPI_IDENTIFY_POPULATION

15.43.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_identify_population {
                BioAPI_IDENTIFY_POPULATION_TYPE Type;
                union {
                BioAPI_DB_HANDLE *BIRDataBase;
                BioAPI_BIR_ARRAY_POPULATION *BIRArray;
                } BIRs;
        } BioAPI_IDENTIFY_POPULATION;
```

15.43.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-IDENTIFY-POPULATION ::= SEQUENCE {
                birs                            CHOICE {
                        birDataBase             BioAPI-DB-HANDLE,
                        birArray                BioAPI-BIR-ARRAY-POPULATION,
                        birPresetArray          NULL
                }
        }
```

15.43.3 Conversion from the pair of C members **Type**/**BIRs** to the ASN.1 component **birs** shall be done by using Table 28 to perform the following actions (in order):

a) determine which alternative of the member **BIRs** is present, based on the value of **Type**;

b) select the alternative of the component **birs** corresponding to the value of **Type**;

c) convert the C member to the alternative of the component **birs**.

15.43.4 Conversion from the ASN.1 component **birs** to the pair of C members **Type**/**BIRs** shall be done by using Table 28 to perform the following actions (in order):

a) set the member **Type** based on the alternative of the component **birs** that is present;

b) select the corresponding alternative of the member **BIRs**;

c) convert the ASN.1 component to the C member.

Table 28 – Mapping between alternatives of the member BIR of the C type BioAPI_IDENTIFY_POPULATION and alternatives of the component bir of the ASN.1 type BioAPI-IDENTIFY-POPULATION

Value of the member Type	Alternative of the member BIRs	Alternative of the component birs	References
1 (BioAPI_DB_TYPE)	BIRDataBase	birDataBase	clause 19 in conjunction with 15.26
2 (BioAPI_ARRAY_TYPE)	BIRArray	birArray	clause 19 in conjunction with 15.7
3 (BioAPI_PRESET_ARRAY_TYPE)	BIRArray	birPresetArray	15.43.5 and 15.43.6
other values	*none – the C value is unconvertible and clause 33 applies*		

15.43.5 Conversion from the C member **BIRArray** to the ASN.1 component **birPresetArray** shall be done by setting the component **birPresetArray** to **NULL**. Any value of **BIRArray** other than **NULL** is unconvertible, and clause 33 applies when such a value is encountered.

15.43.6 Conversion from the ASN.1 component **birPresetArray** to the C member **BIRArray** shall be done by setting the member **BIRArray** to **NULL**.

15.44 Type BioAPI_IDENTIFY_POPULATION_TYPE

15.44.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_IDENTIFY_POPULATION_TYPE;
```

15.44.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define  BioAPI_DB_TYPE                (1)
#define  BioAPI_ARRAY_TYPE             (2)
#define  BioAPI_PRESET_ARRAY_TYPE      (3)
```

15.44.3 There is no corresponding ASN.1 type in BIP.

NOTE – This C type is used in the C types **BioAPI_CANDIDATE** (see 15.20) and **BioAPI_IDENTIFY_POPULATION** (see 15.43) and selects one of three ways of specifying a BIR population in an identification operation. In the corresponding ASN.1 types **BioAPI-CANDIDATE** and **BioAPI-IDENTIFY-POPULATION**, the choice among the three ways of specifying the BIR population is represented by a **CHOICE** construct.

15.45 Type BioAPI_INDICATOR_STATUS

15.45.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_INDICATOR_STATUS;
```

15.45.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define  BioAPI_INDICATOR_ACCEPT       (1)
#define  BioAPI_INDICATOR_REJECT       (2)
#define  BioAPI_INDICATOR_READY        (3)
#define  BioAPI_INDICATOR_BUSY         (4)
#define  BioAPI_INDICATOR_FAILURE      (5)
```

15.45.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-INDICATOR-STATUS ::= ENUMERATED {
                accept,
                reject,
                ready,
                busy,
                failure,
                ...
        }
```

15.45.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done in accordance with Table 29.

Table 29 – Mapping between individual values of the C type BioAPI_INDICATOR_STATUS and abstract values of the ASN.1 type BioAPI-INDICATOR-STATUS

Value of the C type	Abstract value of the ASN.1 type
1 (BioAPI_INDICATOR_ACCEPT)	accept
2 (BioAPI_INDICATOR_REJECT)	reject
3 (BioAPI_INDICATOR_READY)	ready
4 (BioAPI_INDICATOR_BUSY)	busy
5 (BioAPI_INDICATOR_FAILURE)	failure
other values	*none – the C value is unconvertible and clause 33 applies*

15.46 Type BioAPI_INPUT_BIR

15.46.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_input_bir {
        BioAPI_INPUT_BIR_FORM Form;
        union {
        BioAPI_DBBIR_ID *BIRinDb;
        BioAPI_BIR_HANDLE *BIRinBSP;
        BioAPI_BIR *BIR;
        } InputBIR;
} BioAPI_INPUT_BIR;
```

15.46.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-INPUT-BIR ::= SEQUENCE {
        inputBIR                CHOICE {
                birInDB                 BioAPI-DBBIR-ID,
                birInBSP                BioAPI-BIR-HANDLE,
                bir                     BioAPI-BIR
        }
}
```

15.46.3 Conversion from the pair of C members **Form**/**InputBIR** to the ASN.1 component **inputBIR** shall be done by using Table 30 to perform the following actions (in order):

a) determine which alternative of the member **InputBIR** is present, based on the value of **Form**;

b) select the alternative of the component **inputBIR** corresponding to the value of **Form**;

c) convert the C member to the alternative of the component **inputBIR**.

15.46.4 Conversion from the ASN.1 component **inputBIR** to the pair of C members **Form**/**InputBIR** shall be done by using Table 30 to perform the following actions (in order):

a) set the member **Form** based on the alternative of the component **inputBIR** that is present;

b) select the corresponding alternative of the member **InputBIR**;

c) convert the ASN.1 component to the C member.

Table 30 – Mapping between alternatives of the member BIR of the C type BioAPI_INPUT_BIR and alternatives of the component bir of the ASN.1 type BioAPI-INPUT-BIR

Value of the member Form	Alternative of the member InputBIR	Alternative of the component inputBIR	References
1 (BioAPI_DATABASE_ID_INPUT)	BIRinDb	birInDB	clause 19 in conjunction with 15.27
2 (BioAPI_BIR_HANDLE_INPUT)	BIRinBSP	birInBSP	clause 19 in conjunction with 15.12
2 (BioAPI_BIR_HANDLE_INPUT)	BIR	bir	clause 19 in conjunction with 15.12
other values	*none – the C value is unconvertible and clause 33 applies*		

15.47 Type BioAPI_INPUT_BIR_FORM

15.47.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_INPUT_BIR_FORM;
```

15.47.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define  BioAPI_DATABASE_ID_INPUT      (1)
#define  BioAPI_BIR_HANDLE_INPUT       (2)
#define  BioAPI_FULLBIR_INPUT          (3)
```

15.47.3 There is no corresponding ASN.1 type in BIP.

NOTE – This C type is used in the C type **BioAPI_INPUT_BIR** (see 15.46) and selects one of three ways of specifying an input BIR in many operations. In the corresponding ASN.1 type **BioAPI-INPUT-BIR**, the choice among the three ways of specifying the input BIR is represented by a **CHOICE** construct.

15.48 Type BioAPI_OPERATIONS_MASK

15.48.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_OPERATIONS_MASK;
```

15.48.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_ENABLEEVENTS                 (0x00000001)
#define BioAPI_SUBSCRIBETOGUIEVENTS         (0x00000002)
#define BioAPI_CAPTURE                      (0x00000004)
#define BioAPI_CREATETEMPLATE               (0x00000008)
#define BioAPI_PROCESS                      (0x00000010)
#define BioAPI_PROCESSWITHAUXBIR            (0x00000020)
#define BioAPI_VERIFYMATCH                  (0x00000040)
#define BioAPI_IDENTIFYMATCH                (0x00000080)
#define BioAPI_ENROLL                       (0x00000100)
#define BioAPI_VERIFY                       (0x00000200)
#define BioAPI_IDENTIFY                     (0x00000400)
#define BioAPI_IMPORT                       (0x00000800)
#define BioAPI_PRESETIDENTIFYPOPULATION     (0x00001000)
#define BioAPI_DATABASEOPERATIONS           (0x00002000)
#define BioAPI_SETPOWERMODE                 (0x00004000)
#define BioAPI_SETINDICATORSTATUS           (0x00008000)
#define BioAPI_GETINDICATORSTATUS           (0x00010000)
#define BioAPI_CALIBRATESENSOR              (0x00020000)
#define BioAPI_UTILITIES                    (0x00040000)
#define BioAPI_QUERYUNITS                   (0x00100000)
#define BioAPI_QUERYBFPS                    (0x00200000)
#define BioAPI_CONTROLUNIT                  (0x00400000)
```

15.48.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-OPERATIONS-MASK ::= BIT STRING {
        enableEvents                (0),
        subscribeToGUIEvents        (1),
        capture                     (2),
        createTemplate              (3),
        process                     (4),
        processWithAuxBir           (5),
        verifyMatch                 (6),
        identifyMatch               (7),
        enroll                      (8),
        verify                      (9),
        identify                    (10),
        import                      (11),
        presetIdentifyPopulation    (12),
        databaseOperations          (13),
        setPowerMode                (14),
        setIndicatorStatus          (15),
        getIndicatorStatus          (16),
        calibrateSensor             (17),
        utilities                   (18),
        queryUnits                  (20),
        queryBFPs                   (21),
```

```
        controlUnit                     (22)
    } (SIZE(32))
```

15.48.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as follows: Each bit of the ASN.1 bit string shall be mapped to a bit of the C unsigned integer. The leading bit of the bit string (bit 0) shall be mapped to the least significant bit of the unsigned integer (corresponding to the value 0x00000001), and the remaining thirty-one bits shall be mapped in order.

15.49 Type BioAPI_OPTIONS_MASK

15.49.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_OPTIONS_MASK;
```

15.49.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_RAW                          (0x00000001)
#define BioAPI_QUALITY_RAW                  (0x00000002)
#define BioAPI_QUALITY_INTERMEDIATE         (0x00000004)
#define BioAPI_QUALITY_PROCESSED            (0x00000008)
#define BioAPI_APP_GUI                      (0x00000010)
#define BioAPI_GUI_PROGRESS_EVENTS          (0x00000020)
#define BioAPI_SOURCEPRESENT                (0x00000040)
#define BioAPI_PAYLOAD                      (0x00000080)
#define BioAPI_BIR_SIGN                     (0x00000100)
#define BioAPI_BIR_ENCRYPT                  (0x00000200)
#define BioAPI_TEMPLATEUPDATE               (0x00000400)
#define BioAPI_ADAPTATION                   (0x00000800)
#define BioAPI_BINNING                      (0x00001000)
#define BioAPI_SELFCONTAINEDDEVICE          (0x00002000)
#define BioAPI_MOC                          (0x00004000)
#define BioAPI_SUBTYPE_TO_CAPTURE           (0x00008000)
#define BioAPI_SENSORBFP                    (0x00010000)
#define BioAPI_ARCHIVEBFP                   (0x00020000)
#define BioAPI_COMPARISONBFP                (0x00040000)
#define BioAPI_PROCESSINGBFP                (0x00080000)
#define BioAPI_COARSESCORES                 (0x00100000)
```

15.49.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-OPTIONS-MASK ::= BIT STRING {
        raw                     (0),
        qualityRaw              (1),
        qualityIntermediate     (2),
        qualityProcessed        (3),
        appGui                  (4),
        guiProgressEvents       (5),
        sourcePresent           (6),
        payload                 (7),
        birSign                 (8),
        birEncrypt              (9),
        templateUpdate          (10),
        adaptation              (11),
        binning                 (12),
        selfContainedDevice     (13),
        moc                     (14),
        subtypeToCapture        (15),
        sensorBFP               (16),
        archiveBFP              (17),
        comparisonBFP           (18),
        processingBFP           (19),
        coarseScores            (20)
    } (SIZE(32))
```

15.49.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as follows: Each bit of the ASN.1 bit string shall be mapped to a bit of the C unsigned integer. The leading bit of the bit string (bit 0) shall be mapped to the least significant bit of the unsigned integer (corresponding to the value 0x00000001), and the remaining thirty-one bits shall be mapped in order.

15.50 Type BioAPI_POWER_MODE

15.50.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_POWER_MODE;
```

15.50.2 The following manifest constants are defined in BioAPI in support of this C type:

```
#define BioAPI_POWER_NORMAL     (1)
#define BioAPI_POWER_DETECT     (2)
#define BioAPI_POWER_SLEEP      (3)
```

15.50.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-POWER-MODE ::= ENUMERATED {
                normal,
                detect,
                sleep,
                ...
        }
```

15.50.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done in accordance with Table 31.

Table 31 – Mapping between individual values of the C type BioAPI_POWER_MODE and abstract values of the ASN.1 type BioAPI-POWER-MODE

Value of the C type	Abstract value of the ASN.1 type
1 (BioAPI_POWER_NORMAL)	normal
2 (BioAPI_POWER_DETECT)	detect
3 (BioAPI_POWER_SLEEP)	sleep
other values	*none – the C value is unconvertible and clause 33 applies*

15.51 Type BioAPI_QUALITY

15.51.1 This C type is defined in BioAPI as follows:

```
typedef int8_t BioAPI_QUALITY;
```

15.51.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-QUALITY ::= INTEGER (-2..100)
```

15.51.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done as specified in 15.1.2.

15.51.4 Quality values that are outside the range –2 to 100 are unconvertible. Clause 33 applies when such a value is encountered.

15.52 Type BioAPI_RETURN

15.52.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_RETURN;
```

15.52.2 Many manifest constants are defined in BioAPI in support of this C type, most of which indicate error conditions.

15.52.3 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-RETURN ::= UnsignedInt
```

15.52.4 Conversion between the C type and the ASN.1 type (in both directions) shall be done as specified in 15.1.5.

15.53 Type BioAPI_STRING

15.53.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_STRING [269];
```

15.53.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-STRING ::= UTF8String (CONSTRAINED BY
                    {--The UTF-8 encoding shall not contain any NULL characters--
                     --and shall be no longer than 268 octets--})
```

15.53.3 Conversion from the C type to the ASN.1 type shall be done as follows: The content of the octet array pointed to by the C variable up to the first zero-valued octet (exclusive) shall be interpreted as the UTF-8 encoding of a character string. The ASN.1 abstract value shall be set to that character string.

15.53.4 Conversion from the ASN.1 type to the C type shall be done as follows: Calling *L* the length (in octets) of the UTF-8 encoding of the ASN.1 abstract character string, the first *L* + 1 octets of the octet array pointed to by the C variable shall be filled with that UTF-8 encoding followed by a zero-valued octet.

NOTE – The constraint present on the ASN.1 type definition guarantees that the C octet array will not overflow during this conversion.

15.54 Type BioAPI_TIME

15.54.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_time {
        uint8_t Hour;
        uint8_t Minute;
        uint8_t Second;
} BioAPI_TIME;
```

15.54.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-TIME ::= SEQUENCE {
        hour            INTEGER (0..99),
        minute          INTEGER (0..99),
        second          INTEGER (0..99)
}
```

15.54.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 32.

Table 32 – Mapping between members of the C type BioAPI_TIME and components of the ASN.1 type BioAPI-TIME

Member of the C type	Component of the ASN.1 type	References
Hour	hour	15.1.2
Minute	minute	15.1.2
Second	second	15.1.2

15.54.4 The abstract value of type **BioAPI-TIME** in which the three components **hour**, **minute**, and **second** have the value 99 is valid. Apart from that particular abstract value, all **hour** values outside the range 0 to 23, all **minute** values outside the range 0 to 59, and all **second** values outside the range 0 to 59 are unconvertible, and clause 33 applies when such a value is encountered.

15.55 Type BioAPI_UNIT_ID

15.55.1 This C type is defined in BioAPI as follows:

```
typedef uint32_t BioAPI_UNIT_ID;
```

15.55.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-UNIT-ID ::= UnsignedInt
```

15.55.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done as specified in 15.1.5.

15.56 Type BioAPI_UNIT_LIST_ELEMENT

15.56.1 This C type is defined in BioAPI as follows:

```
typedef struct _bioapi_unit_list_element     {
                BioAPI_CATEGORY UnitCategory;
                BioAPI_UNIT_ID UnitID;
        } BioAPI_UNIT_LIST_ELEMENT;
```

15.56.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-UNIT-LIST-ELEMENT ::= SEQUENCE {
                category        BioAPI-CATEGORY,
                unitID          BioAPI-UNIT-ID
        }
```

15.56.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 33.

Table 33 – Mapping between members of the C type BioAPI_UNIT_LIST_ELEMENT and components of the ASN.1 type BioAPI-UNIT-LIST-ELEMENT

Member of the C type	Component of the ASN.1 type	References
UnitCategory	category	15.21
UnitID	unitID	15.55

15.57 Type BioAPI_UNIT_SCHEMA

15.57.1 This C type is defined in BioAPI as follows:

```
typedef struct bioapi_unit_schema {
                BioAPI_UUID             BSPProductUuid;
                BioAPI_UUID             UnitManagerProductUuid;
                BioAPI_UNIT_ID          UnitId;
                BioAPI_CATEGORY         UnitCategory;
                BioAPI_UUID             UnitProperties;
                BioAPI_STRING           VendorInformation;
                BioAPI_EVENT_MASK       SupportedEvents;
                BioAPI_UUID             UnitPropertyUuid;
                BioAPI_DATA             UnitProperty;
                BioAPI_STRING           HardwareVersion;
                BioAPI_STRING           FirmwareVersion;
                BioAPI_STRING           SoftwareVersion;
                BioAPI_STRING           HardwareSerialNumber;
                BioAPI_BOOL             AuthenticatedHardware;
                uint32_t                MaxBSPDbSize;
                uint32_t                MaxIdentify;
        } BioAPI_UNIT_SCHEMA;
```

15.57.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-UNIT-SCHEMA ::= SEQUENCE {
                bspProductUuid          BioAPI-UUID,
                unitManagerProductUuid  BioAPI-UUID,
                unitId                  BioAPI-UNIT-ID,
                category                BioAPI-CATEGORY,
                unitProperties          BioAPI-UUID,
                vendorInformation       BioAPI-STRING,
                supportedUnitEvents     BioAPI-UNIT-EVENT-TYPE-MASK,
                propertyUuid            BioAPI-UUID,
                property                BioAPI-DATA,
                hardwareVersion         BioAPI-STRING,
                firmwareVersion         BioAPI-STRING,
                softwareVersion         BioAPI-STRING,
                hardwareSerialNumber    BioAPI-STRING,
                authenticatedHardware   BOOLEAN,
                maxBSPDbSize            UnsignedInt,
                maxIdentify             UnsignedInt
        }
```

15.57.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done by converting between individual C members and ASN.1 components in accordance with Table 34.

Table 34 – Mapping between members of the C type BioAPI_UNIT_SCHEMA and components of the ASN.1 type BioAPI-UNIT-SCHEMA

Member of the C type	Component of the ASN.1 type	References
BSPProductUuid	bspProductUuid	15.58
UnitManagerProductUuid	unitManagerProductUuid	15.58
UnitId	unitId	15.55
UnitCategory	category	15.21
UnitProperties	unitProperties	15.58
VendorInformation	vendorInformation	15.53
SupportedEvents	supportedUnitEvents	15.31
UnitPropertyUuid	propertyUuid	15.58
UnitProperty	property	15.22
HardwareVersion	hardwareVersion	15.53
FirmwareVersion	firmwareVersion	15.53
SoftwareVersion	softwareVersion	15.53
HardwareSerialNumber	hardwareSerialNumber	15.53
AuthenticatedHardware	authenticatedHardware	15.18
MaxBSPDbSize	maxBSPDbSize	15.1.5
MaxIdentify	maxIdentify	15.1.5

NOTE – No "**HostingEndpointIRI**" parameter or BSP access UUID parameter is present in the C type **BioAPI_UNIT_SCHEMA**. All the units returned by **BioAPI_QueryUnits** are in the same BIP endpoint, which is the hosting endpoint of the BSP. The unit schema provided by a **BioAPI_EVENT_HANDLER** function belongs to a unit that is managed (directly or indirectly) by the BSP whose UUID is provided as the first parameter of that function.

15.58 Type BioAPI_UUID

15.58.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_UUID[16];
```

15.58.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-UUID ::= OCTET STRING (SIZE(16))
```

15.58.3 Conversion from the C type to the ASN.1 type shall be done by setting the ASN.1 abstract value to an octet string consisting of the 16 octets in the octet array pointed to by the C variable.

15.58.4 Conversion from the ASN.1 type to the C type shall be done by filling the octet array pointed to by the C variable with the 16 octets in the ASN.1 octet string.

15.59 Type BioAPI_VERSION

15.59.1 This C type is defined in BioAPI as follows:

```
typedef uint8_t BioAPI_VERSION;
```

15.59.2 The corresponding ASN.1 type in BIP is defined as follows:

```
BioAPI-VERSION ::= SEQUENCE {
        major       INTEGER (0..15),
        minor       INTEGER (0..15)
    }
```

15.59.3 Conversion between the C type and the ASN.1 type (in both directions) shall be done as follows: The most significant half octet and the least significant half octet of the C integer shall be treated as two separate (integer) members of the C type, and each member shall be converted in accordance with Table 35.

Table 35 – Mapping between members of the C type BioAPI_VERSION and components of the ASN.1 type BioAPI-VERSION

Member of the C type	Component of the ASN.1 type	References
Most significant half octet	major	15.1.2
Least significant half octet	minor	15.1.2

16 Functions defined in BioAPI and corresponding BIP messages

16.1 Function BioAPI_Init

16.1.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Init
        (BioAPI_VERSION Version);
```

16.1.2 There are no related BIP message types.

16.1.3 When a framework receives a call to the function **BioAPI_Init** from the local application, it shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create all the conceptual tables initially empty;

d) set the endpoint IRI of the local endpoint to a valid unique IRI;

NOTE 1 – The means by which a framework chooses an IRI are not specified in this Recommendation | International Standard.

e) add an entry to the **VisibleEndpoints** table (see 18.2), where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI; and

2) the remaining components shall be set from the attributes of the framework schema in the local component registry;

f) for each BSP schema in the local component registry, add an entry to the **VisibleBSPRegistrations** table (see 18.3), where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

2) the component **bspAccessUuid** shall be set to a dynamically generated UUID; and

3) the remaining components shall be set from the attributes of the BSP schema in the local component registry;

NOTE 2 – A different BSP access UUID may be generated for the same BSP each time **BioAPI_Init** or **BIOAPI_InitEndpoint** is called by the local application.

g) for each BFP schema in the local component registry, add an entry to the **VisibleBFPRegistrations** table (see 18.4), where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI; and

2) the remaining components shall be set from the attributes of the BFP schema in the local component registry;

h) optionally, behave as if the framework had received one or more calls to the function **BioAPI_LinkToEndpoint** from the local application (see 16.4), with the **slaveEndpointIRI** parameter value based on framework configuration data;

i) return the value 0 to the local application.

NOTE 3 – The means by which the framework determines the set of BIP endpoints to link to during a **BioAPI_Init** function call are not specified in this Recommendation | International Standard. This information might be obtained from configuration parameters of the framework implementation.

16.2 Function BioAPI_InitEndpoint

16.2.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_InitEndpoint
        (BioAPI_VERSION Version,
        const uint8_t *LocalEndpointIRI);
```

16.2.2 There are no related BIP message types.

16.2.3 When a framework receives a call to the function **BioAPI_InitEndpoint** from the local application, it shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create all the conceptual tables initially empty;

d) if the value of **LocalEndpointIRI** is not **NULL**, then set the endpoint IRI of the local endpoint to that value;

e) if the value of **LocalEndpointIRI** is **NULL**, then either set the endpoint IRI of the local endpoint to any valid unique IRI, or return the value **BioAPIERR_LOCAL_ENDPOINT_IRI_NEEDED** to the local application, skipping the remaining actions;

> NOTE 1 – The means by which a framework chooses an IRI are not specified in this Recommendation | International Standard. In addition, a framework that acts as a slave needs the ability to set the local endpoint IRI even in the absence of a local application that makes a call to call to **BioAPI_InitEndpoint**.

f) add an entry to the **VisibleEndpoints** table (see 18.2), where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI; and

2) the remaining components shall be set from the attributes of the framework schema in the local component registry;

g) for each BSP schema in the local component registry, add an entry to the **VisibleBSPRegistrations** table (see 18.3), where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

2) the component **bspAccessUuid** shall be set to a dynamically generated UUID; and

3) the remaining components shall be set from the attributes of the BSP schema in the local component registry;

> NOTE 2 – A different BSP access UUID may be generated for the same BSP each time **BioAPI_Init** or **BioAPI_InitEndpoint** is called by the local application.

h) for each BFP schema in the local component registry, add an entry to the **VisibleBFPRegistrations** table (see 18.4), where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI; and

2) the remaining components shall be set from the attributes of the BFP schema in the local component registry;

i) optionally, behave as if the framework had received one or more calls to the function **BioAPI_LinkToEndpoint** from the local application (see 16.4), with the **slaveEndpointIRI** parameter value based on framework configuration data;

j) return the value 0 to the local application.

> NOTE 3 – The means by which the framework determines the set of BIP endpoints to link to during a **BioAPI_InitEndpoint** function call are not specified in this Recommendation | International Standard. This information might be obtained from configuration parameters of the framework implementation.

16.3 Function BioAPI_Terminate

16.3.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Terminate
        (void);
```

16.3.2 One BIP message type is related to this function: the **masterDeletionEvent** notification BIP message type. This BIP message type carries a value of the following BIP message parameter ASN.1 type:

```
MasterDeletionEvent-NotificationParams ::= NULL
```

16.3.3 When a framework receives a call to the function **BioAPI_Terminate** from the local application, it shall perform the actions specified in the five following subclauses (in order).

16.3.3.1 For each entry of the **VisibleEndpoints** table (see 18.2) where the component **hostingEndpointIRI** (say, *slaveEndpointIRI*) has a value different from the local endpoint IRI, the framework shall perform the following actions (in order):

a) create and send a **deleteMaster** request BIP message (see 13.1 and 16.5.2) with the slave endpoint IRI set to *slaveEndpointIRI* and the parameter value set to **NULL**;

b) receive a corresponding **deleteMaster** response BIP message (see 13.6); and

c) delete the entry of the framework schema table (subclause 18.2.3 applies);

NOTE – Those slave endpoints are all the BIP endpoints to which an **addMaster** request BIP message has been sent but no subsequent **deleteMaster** request BIP message has been sent. This includes any **addMaster** request BIP messages sent while processing a **BioAPI_Init** or **BioAPI_InitEndpoint** function call (see 16.1).

d) optionally, destroy the transport-level connection(s) with the slave endpoint as required by the transport protocol binding in use.

16.3.3.2 For each entry (say, *masterEndpoint*) of the **MasterEndpoints** table (see 18.1), the framework shall perform the following actions (in order):

a) create and send a **masterDeletionEvent** notification BIP message (see 13.4) with the master endpoint IRI set from the component **masterEndpointIRI** of *masterEndpoint* and the parameter value set to **NULL**;

b) delete the entry of the **MasterEndpoints** table (subclause 18.1.3 applies);

c) optionally, destroy the transport-level connection(s) with the master endpoint as required by the transport protocol binding in use.

16.3.3.3 Delete all the entries of the **ApplicationOwnedMemoryBlocks** table (see 18.13).

16.3.3.4 Make an internal BioAPI function call (see 13.10) to the same function as the incoming call, and return to the local application the return value of the internal call.

16.3.3.5 Delete all temporary abstract values that have not yet been deleted.

16.3.4 When a framework receives (see 13.9) a **masterDeletionEvent** notification BIP message from a slave endpoint, it shall perform the following actions (in order):

a) delete the entry (if any) of the **VisibleEndpoints** table where the component **hostingEndpointIRI** contains the slave endpoint IRI (subclause 18.2.3 applies);

b) conclude that no acknowledgement BIP message is to be sent.

16.4 Function BioAPI_LinkToEndpoint

16.4.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI_LinkToEndpoint
    (const uint8_t *SlaveEndpointIRI);
```

16.4.2 A pair of BIP message types are related to this function: the **addMaster** request BIP message type and the **addMaster** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
AddMaster-RequestParams ::= SEQUENCE {
        bipVersion              BioAPI-VERSION
    }
```

and:

```
AddMaster-ResponseParams ::= SEQUENCE {
        fwSchema                BioAPI-FRAMEWORK-SCHEMA OPTIONAL,
        bspSchemas              SEQUENCE (SIZE(0..max-unsigned-int)) OF
                                    bspSchema BioAPI-BSP-SCHEMA,
        bfpSchemas              SEQUENCE (SIZE(0..max-unsigned-int)) OF
                                    bfpSchema BioAPI-BFP-SCHEMA
    }
```

16.4.3 The following ASN.1 type is defined to aid the specification of the behaviour of a framework, but its abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
LinkCallParams ::= SEQUENCE {
        slaveEndpointIRI    EndpointIRI
    }
```

16.4.4 When a framework receives a call to the function **BioAPI_LinkToEndpoint** from the local application, it shall perform the following actions (in order):

a) set the *link number* associated with the BIP link to a randomly generated value and the *request identifier* associated with the BIP link to another randomly generated value;

NOTE – The link number is not, strictly speaking, an identifier of the link, because there is a (small) probability of duplication. The main purposes of the link number are to increase the robustness of implementations and to facilitate diagnostics.

b) create a temporary abstract value (say, *linkCallParams*) of type **LinkCallParams** (see 16.4.3) by converting from the parameters of the **BioAPI_LinkToEndpoint** function call as specified in 16.4.6;

c) search the **VisibleEndpoints** table (see 18.2) for an entry where the component **hostingEndpointIRI** has the same value as the component **slaveEndpointIRI** of *linkCallParams*;

d) optionally, establish one or two transport-level connections with the slave endpoint as required by the transport protocol binding(s) in use;

e) if such an entry is found, then return the value **BioAPIERR_SLAVE_ALREADY_LINKED** to the local application, skipping the remaining actions;

f) create a temporary abstract value (say, *outgoingRequestParams*) of type **AddMaster-RequestParams** (see 16.4.2), where the component **bipVersion** shall be set to the version number associated with this edition of this Recommendation | International Standard (see 13.15);

g) create and send an **addMaster** request BIP message (see 13.2) with the slave endpoint IRI set from the component **slaveEndpointIRI** of *linkCallParams* and the parameter value set to *outgoingRequestParams*;

h) receive a corresponding **addMaster** response BIP message (see 13.6);

i) if the return value of the **addMaster** response BIP message is not 0, then return that value to the local application, skipping the remaining actions;

j) let *incomingResponseParams* be the parameter value (of type **AddMaster-ResponseParams** – see 16.4.2) of the **addMaster** response BIP message;

k) if the optional component **fwSchema** is absent, then return the value **BioAPIERR_FRAMEWORK_SCHEMA_ABSENT** to the local application, skipping the remaining actions;

l) if the same BSP product UUID occurs as the value of the component **bspProductUuid** in two or more elements of the component **bspSchemas** of *incomingResponseParams*, then return the value **BioAPIERR_DUPLICATE_BSP_PRODUCT_UUID** to the local application, skipping the remaining actions;

m) if the same BFP product UUID occurs as the value of the component **bfpProductUuid** in two or more elements of the component **bfpSchemas** of *incomingResponseParams*, then return the value **BioAPIERR_DUPLICATE_BFP_PRODUCT_UUID** to the local application, skipping the remaining actions;

n) add an entry to the **VisibleEndpoints** table (see 18.2), where:

   1) the component **hostingEndpointIRI** shall be set from the component **slaveEndpointIRI** of *linkCallParams*; and

   2) the remaining components shall be set from the components of the component **fwSchema** of *incomingResponseParams*;

o) for each element (say, *bspSchema*) of the component **bspSchemas** of *incomingResponseParams*, add an entry to the **VisibleBSPRegistrations** table (see 18.3), where:

   1) the component **hostingEndpointIRI** shall be set from the component **slaveEndpointIRI** of *linkCallParams*;

   2) the component **bspAccessUuid** shall be set to a dynamically generated UUID; and

   3) the remaining components shall be set from the components of *bspSchema*;

p) for each element (say, *bfpSchema*) of the component **bfpSchemas** of *incomingResponseParams*, add an entry to the **VisibleBFPRegistrations** table (see 18.4), where:

1) the component **hostingEndpointIRI** shall be set from the component **slaveEndpointIRI** of *linkCallParams*; and

2) the remaining components shall be set from the components of *bfpSchema*;

q) return the value 0 to the local application.

16.4.5 When a framework receives (see 13.9) an **addMaster** request BIP message from a BIP endpoint, it shall perform the following actions (in order):

a) set the *link number* associated with the BIP link to the link number found in the incoming request BIP message and set the *notification identifier* associated with the BIP link to another randomly generated value;

b) optionally, create and send a corresponding **addMaster** response BIP message (see 13.3) with the return value set to **BioAPIERR_UNWILLING_TO_ACT_AS_SLAVE**, skipping the remaining actions;

NOTE – This indicates to the sending endpoint that this BIP endpoint is rejecting an endpoint linking request. The criteria for accepting or rejecting an endpoint linking request are not specified in this Recommendation | International Standard.

c) if the component **bipVersion** of the parameter value of the **addMaster** request BIP message has a value different from the version number associated with this Recommendation | International Standard (see 13.15), then the framework shall either:

1) create and send a corresponding **addMaster** response BIP message (see 13.3) with the return value set to **BioAPIERR_INCOMPATIBLE_VERSION**, skipping the remaining actions; or

2) perform any actions that are appropriate for that version number (but are not specified in this Recommendation | International Standard), skipping the remaining actions;

d) search the **MasterEndpoints** table (see 18.1) for an entry where the component **masterEndpointIRI** contains the endpoint IRI of the sending BIP endpoint;

e) if such an entry is found, then create and send a corresponding **addMaster** response BIP message (see 13.3) with the return value set to **BioAPIERR_MASTER_ALREADY_LINKED**, skipping the remaining actions;

f) add an entry to the **MasterEndpoints** table (see 18.1), where the component **masterEndpointIRI** shall be set to the endpoint IRI of the sending BIP endpoint;

g) locate the entry (say, *fwSchema*) of the **VisibleEndpoints** table (see 18.2) where the component **hostingEndpointIRI** contains the local endpoint IRI;

h) create a temporary abstract value (say, *outgoingResponseParams*) of type **AddMaster-ResponseParams** (see 16.4.2), where the component **fwSchema** shall be present and shall be set to *fwSchema*, and the components **bspSchemas** and **bfpSchemas** shall be initially empty;

i) for each entry (say, *bspSchema*) of the **VisibleBSPRegistrations** table (see 18.3) where the component **hostingEndpointIRI** contains the local endpoint IRI, add an element to the component **bspSchemas** of *outgoingResponseParams*, where:

1) the component **bspAccessUuid** shall be set to sixteen zero-valued octets; and

2) the remaining components of the element shall be set from the components of *bspSchema* with the same names;

j) for each entry (say, *bfpSchema*) of the **VisibleBFPRegistrations** table (see 18.4) where the component **hostingEndpointIRI** contains the local endpoint IRI, add an element to the component **bfpSchemas** of *outgoingResponseParams*, where all the components of the element shall be set from the components of *bfpSchema* with the same names;

k) create and send a corresponding **addMaster** response BIP message (see 13.3) with the parameter value set to *outgoingResponseParams* and the return value set to 0.

16.4.6 Conversion from the parameters of the C function **BioAPI_LinkToEndpoint** to the ASN.1 type **LinkCallParams** (see 16.4.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 36.

Table 36 – Mapping from the parameters of the function BioAPI_LinkToEndpoint to the ASN.1 type LinkCallParams

Function parameter	Component of the ASN.1 type	References
SlaveEndpointIRI	slaveEndpointIRI	15.3

16.5 Function BioAPI_UnlinkFromEndpoint

16.5.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI_UnlinkFromEndpoint
        (const uint8_t *SlaveEndpointIRI);
```

16.5.2 A pair of BIP message types are related to this function: the **deleteMaster** request BIP message type and the **deleteMaster** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
DeleteMaster-RequestParams ::= NULL
```

and:

```
DeleteMaster-ResponseParams ::= NULL
```

16.5.3 The following ASN.1 type is defined to aid the specification of the behaviour of a framework, but its abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
UnlinkCallParams ::= SEQUENCE {
        slaveEndpointIRI    EndpointIRI
    }
```

16.5.4 When a framework receives a call to the function **BioAPI_UnlinkFromEndpoint** from the local application, it shall perform the following actions (in order):

a) create a temporary abstract value (say, *unlinkCallParams*) of type **UnlinkCallParams** (see 16.5.3) by converting from the parameters of the **BioAPI_UnlinkFromEndpoint** function call as specified in 16.5.6;

b) let *slaveEndpointIRI* be the value of the component **slaveEndpointIRI** of *unlinkCallParams*;

c) search the **VisibleEndpoints** table (see 18.2) for an entry where the component **hostingEndpointIRI** has the value *slaveEndpointIRI*;

d) if there is no such entry, then return the value **BioAPIERR_NO_SUCH_SLAVE_FOUND** to the local application, skipping the remaining actions;

e) create and send a **deleteMaster** request BIP message (see 13.2) with the slave endpoint IRI set to *slaveEndpointIRI* and the parameter value set to **NULL**;

f) receive a corresponding **deleteMaster** response BIP message (see 13.6);

g) delete the entry of the **VisibleEndpoints** table (subclause 18.2.3 applies);

h) if the return value of the **deleteMaster** response BIP message is not 0, then return that value to the local application, skipping the remaining actions;

i) optionally, destroy the transport-level connection(s) with the slave endpoint as required by the transport protocol binding in use;

j) return 0 to the local application.

16.5.5 When a framework receives (see 13.9) a **deleteMaster** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) search the **MasterEndpoints** table (see 18.1) for an entry where the component **masterEndpointIRI** contains the endpoint IRI of the sending BIP endpoint;

b) if there is no such entry, then create and send a corresponding **deleteMaster** response BIP message (see 13.3) with the return value set to **BioAPIERR_NO_SUCH_MASTER_FOUND**, skipping the remaining actions;

c) delete the entry of the **MasterEndpoints** table (subclause 18.1.3 applies);

d) create and send a corresponding **deleteMaster** response BIP message (see 13.3) with the parameter value set to **NULL** and the return value set to 0;

e) optionally, destroy the transport-level connection(s) with the master endpoint as required by the transport protocol binding in use.

16.5.6 Conversion from the parameters of the C function **BioAPI_UnlinkFromEndpoint** to the ASN.1 type **UnlinkCallParams** (see 16.5.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 37.

Table 37 – Mapping from the parameters of the function BioAPI_UnlinkFromEndpoint to the ASN.1 type UnlinkCallParams

Function parameter	Component of the ASN.1 type	References
SlaveEndpointIRI	slaveEndpointIRI	15.3

16.6 Function BioAPI_EnumFrameworks

16.6.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI_EnumFrameworks
        (BioAPI_FRAMEWORK_SCHEMA **FwSchemaArray,
        unit32_t *NumberOfElements);
```

16.6.2 There are no related BIP message types.

16.6.3 The following ASN.1 type is defined to aid the specification of the behaviour of a framework, but its abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
EnumFrameworksCallOutputParams ::= SEQUENCE OF BioAPI-FRAMEWORK-SCHEMA
```

16.6.4 When a framework receives a call to the function **BioAPI_EnumFrameworks** from the local application, it shall perform the following actions (in order):

a) create a temporary abstract value (say, *outgoingResponseParams*) of type **EnumFrameworksCallOutputParams** (see 16.6.3) initially empty;

b) add each entry of the **VisibleEndpoints** table (see 18.2) to *outgoingResponseParams*;

c) set the output parameters of the **BioAPI_EnumFrameworks** function call by converting from *outgoingResponseParams* as specified in 16.6.5;

d) return the value 0 to the local application.

16.6.5 Conversion from the parameters of the C function **BioAPI_EnumFrameworks** to the ASN.1 type **EnumFrameworksCallOutputParams** (see 16.6.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 38.

Table 38 – Mapping from the ASN.1 type EnumFrameworksCallOutputParams to the parameters of the function BioAPI_EnumFrameworks

Component of the ASN.1 type	Function parameter	References
fwSchemas	FwSchemaArray, NumberOfElements	clause 20 in conjunction with 16.6.6

16.6.6 Conversion from the ASN.1 component **fwSchemas** to the pair of C variables pointed to by the parameters **FwSchemaArray**/**NumberOfElements** shall be done as follows: Calling *N* the number of elements of the component **fwSchemas**, a newly allocated array of *N* elements of type **BioAPI_FRAMEWORK_SCHEMA** (see 15.33) shall be filled by converting each element of the component **fwSchemas**, in order, to an element of the array as specified in 15.33. The C variable pointed to by the parameter **FwSchemaArray** shall be set to the address of the array, and the C variable pointed to by the parameter **NumberOfElements** shall be set to *N*.

16.7 Function BioAPI_EnumBSPs

16.7.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_EnumBSPs
        (BioAPI_BSP_SCHEMA **BSPSchemaArray,
        uint32_t *NumberOfElements);
```

16.7.2 There are no related BIP message types.

16.7.3 The following ASN.1 type is defined to aid the specification of the behaviour of a framework, but its abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
EnumBSPsCallOutputParams ::= SEQUENCE OF BioAPI-BSP-SCHEMA
```

16.7.4 When a framework receives a call to the function **BioAPI_EnumBSPs** from the local application, it shall perform the following actions (in order):

a) create a temporary abstract value (say, *outgoingResponseParams*) of type **EnumBSPsCallOutputParams** (see 16.7.3) initially empty;

b) add each entry of the **VisibleBSPRegistrations** table (see 18.3) to *outgoingResponseParams*;

c) set the output parameters of the **BioAPI_EnumBSPs** function call by converting from *outgoingResponseParams* as specified in 16.7.5;

d) return the value 0 to the local application.

16.7.5 Conversion from the parameters of the C function **BioAPI_EnumBSPs** to the ASN.1 type **EnumBSPsCallOutputParams** (see 16.7.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 39.

Table 39 – Mapping from the ASN.1 type EnumBSPsCallOutputParams to the parameters of the function BioAPI_EnumBSPs

Component of the ASN.1 type	Function parameter	References
bspSchemas	BSPSchemaArray, NumberOfElements	clause 20 in conjunction with 16.7.6

16.7.6 Conversion from the ASN.1 component **bspSchemas** to the pair of C variables pointed to by the parameters **BSPSchemaArray**/**NumberOfElements** shall be done as follows: Calling *N* the number of elements of the component **bspSchemas**, a newly allocated array of *N* elements of type **BioAPI_BSP_SCHEMA** (see 15.19) shall be filled by converting each element of the component **bspSchemas**, in order, to an element of the array as specified in 15.19. The C variable pointed to by the parameter **BSPSchemaArray** shall be set to the address of the array, and the C variable pointed to by the parameter **NumberOfElements** shall be set to *N*.

16.8 Function BioAPI_EnumBFPs

16.8.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_EnumBFPs
        (BioAPI_BFP_SCHEMA **BFPSchemaArray,
        uint32_t *NumberOfElements);
```

16.8.2 There are no related BIP message types.

16.8.3 The following ASN.1 type is defined to aid the specification of the behaviour of a framework, but its abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
EnumBFPsCallOutputParams ::= SEQUENCE OF BioAPI-BFP-SCHEMA
```

16.8.4 When a framework receives a call to the function **BioAPI_EnumBFPs** from the local application, it shall perform the following actions (in order):

a) create a temporary abstract value (say, *outgoingResponseParams*) of type **EnumBFPsCallOutputParams** (see 16.8.3) initially empty;

b) add each entry of the **VisibleBFPRegistrations** table (see 18.4) to *outgoingResponseParams*;

c) set the output parameters of the **BioAPI_EnumBFPs** function call by converting from *outgoingResponseParams* as specified in 16.8.5;

d) return the value 0 to the local application.

16.8.5 Conversion from the parameters of the C function **BioAPI_EnumBFPs** to the ASN.1 type **EnumBFPsCallOutputParams** (see 16.8.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 40.

Table 40 – Mapping from the ASN.1 type EnumBFPsCallOutputParams to the parameters of the function BioAPI_EnumBFPs

Component of the ASN.1 type	Function parameter	References
bfpSchemas	BFPSchemaArray, NumberOfElements	clause 20 in conjunction with 16.8.6

16.8.6 Conversion from the ASN.1 component **bfpSchemas** to the pair of C variables pointed to by the parameters **BFPSchemaArray**/**NumberOfElements** shall be done as follows: Calling *N* the number of elements of the component **bfpSchemas**, a newly allocated array of *N* elements of type **BioAPI_BFP_SCHEMA** (see 15.5) shall be filled by converting each element of the component **bfpSchemas**, in order, to an element of the array as specified in 15.5. The C variable pointed to by the parameter **BFPSchemaArray** shall be set to the address of the array, and the C variable pointed to by the parameter **NumberOfElements** shall be set to *N*.

16.9 Function BioAPI_BSPLoad

16.9.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_BSPLoad
        (const BioAPI_UUID *BSPUuid,
        BioAPI_EVENT_HANDLER EventHandler,
        void* EventHandlerCtx);
```

16.9.2 A pair of BIP message types are related to this function: the **bspLoad** request BIP message type and the **bspLoad** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
BSPLoad-RequestParams ::= SEQUENCE {
        bspProductUuid              BioAPI-UUID,
        unitEventSubscription       BOOLEAN
    }
```

and:

```
BSPLoad-ResponseParams ::= NULL
```

16.9.3 The following ASN.1 type is defined to aid the specification of the behaviour of a framework, but its abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
BSPLoadCallParams ::= SEQUENCE {
        bspUuid                     BioAPI-UUID,
        unitEventHandlerAddress     MemoryAddress,
        unitEventHandlerContext     MemoryAddress
    }
```

16.9.4 When a framework receives a call to the function **BioAPI_BSPLoad** from the local application, it shall first determine the hosting endpoint and the product UUID of the BSP (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23, and then one of the three following subclauses applies.

16.9.4.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *bspLoadCallParams*) of type **BSPLoadCallParams** (see 16.9.3) by converting from the parameters of the **BioAPI_BSPLoad** function call as specified in 16.9;

b) add an entry to the **RunningBSPLocalReferences** table (see 18.5), where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

2) the component **bspProductUuid** shall be set to *bspProductUuid*;

3) if the component **bspUuid** of *bspLoadCallParams* has a value different from *bspProductUuid*, then the component **useBSPAccessUuid** of the entry shall be set to **TRUE**; otherwise, it shall be set to **FALSE**; and

4) the components **unitEventHandlerAddress** and **unitEventHandlerContext** shall be set from the components of *bspLoadCallParams* with the same names;

NOTE – Postponing the internal function call to after the addition of the entry to the table allows notifying to the local application any unit events generated during the internal function call.

c) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPUuid** shall be set by converting from *bspProductUuid* as specified in clause 19 in conjunction with 15.58;

d) if the return value of the internal call is not 0, then delete the entry added above to the **RunningBSPLocalReferences** table and return that value to the local application, skipping the remaining action;

e) return the value 0 to the local application.

16.9.4.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *bspLoadCallParams*) of type **BSPLoadCallParams** (see 16.9.3) by converting from the parameters of the **BioAPI_BSPLoad** function call as specified in 16.9;

b) add an entry to the **RunningBSPLocalReferences** table (see 18.5), where:

1) the component **hostingEndpointIRI** shall be set to the hosting endpoint IRI;

2) the component **bspProductUuid** shall be set to *bspProductUuid*;

3) if the component **bspUuid** of *bspLoadCallParams* has a value different from *bspProductUuid*, then the component **useBSPAccessUuid** of the entry shall be set to **TRUE**; otherwise, it shall be set to **FALSE**; and

4) the components **unitEventHandlerAddress** and **unitEventHandlerContext** shall be set from the components of *bspLoadCallParams* with the same names;

NOTE – Postponing the request to after the addition of the entry to the table allows notifying to the local application any unit events notified by the slave endpoint while processing the request.

c) create a temporary abstract value (say, *outgoingRequestParams*) of type **BSPLoad-RequestParams** (see 16.9.2) where:

1) the component **bspProductUuid** shall be set to *bspProductUuid*; and

2) if the component **unitEventHandlerAddress** of *bspLoadCallParams* has a value different from 0, then the component **unitEventSubscription** of *outgoingRequestParams* shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

d) create and send a **bspLoad** request BIP message (see 13.2) with the slave endpoint IRI set to the hosting endpoint IRI and the parameter value set to *outgoingRequestParams*;

e) receive a corresponding **bspLoad** response BIP message (see 13.6);

f) if the return value of the response BIP message is not 0, then delete the entry added above to the **RunningBSPLocalReferences** table, and return that value to the local application, skipping the remaining actions;

g) return 0 to the local application.

16.9.4.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.9.5 When a framework receives (see 13.9) a **bspLoad** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **BSPLoad-RequestParams** – see 16.9.2) of the **bspLoad** request BIP message;

b) add an entry to the **RunningBSPRemoteReferences** table (see 18.6), where the component **referrerEndpointIRI** shall be set to the master endpoint IRI, and the components **bspProductUuid** and **unitEventSubscription** shall be set from the components of *incomingRequestParams* with the same names;

NOTE – Postponing the internal function call to after the addition of the entry to the table allows notifying to the master endpoint any unit events generated during the internal function call.

c) create a temporary abstract value (say, *bspLoadCallParams*) of type **BSPLoadCallParams** (see 16.9.3), where:

1) the component **bspUuid** shall be set from the component **bspProductUuid** of *incomingRequestParams*; and

2) the components **unitEventHandlerAddress** and **unitEventHandlerContext** shall be set to 0;

d) make an internal BioAPI function call (see 13.10) to the function **BioAPI_BSPLoad**, where the parameters of the function call shall be set by converting from *bspLoadCallParams* as specified in 16.9;

e) if the return value of the internal call is not 0, then delete the entry added above to the **RunningBSPRemoteReferences** table, and create and send a corresponding **bspLoad** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

f) create and send a corresponding **bspLoad** response BIP message (see 13.3) with the parameter value set to *outgoingResponseParams* and the return value set to 0.

16.9.6 Conversion between the parameters of the C function **BioAPI_BSPLoad** and the ASN.1 type **BSPLoadCallParams** (see 16.9.3) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 41.

Table 41 – Mapping between the parameters of the function BioAPI_BSPLoad and the ASN.1 type BSPLoadCallParams

Function parameter	Component of the ASN.1 type	References
BSPUuid	bspUuid	clause 19 in conjunction with 15.58
EventHandler	unitEventHandlerAddress	15.1.7
EventHandlerCtx	unitEventHandlerContext	15.1.7

16.10 Function BioAPI_BSPUnload

16.10.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_BSPUnload
        (const BioAPI_UUID *BSPUuid,
        BioAPI_EVENT_HANDLER EventHandler,
        void* EventHandlerCtx);
```

16.10.2 A pair of BIP message types are related to this function: the **bspUnload** request BIP message type and the **bspUnload** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
BSPUnload-RequestParams ::= SEQUENCE {
        bspProductUuid              BioAPI-UUID,
        unitEventSubscription       BOOLEAN
    }
```

and:

```
BSPUnload-ResponseParams ::= NULL
```

16.10.3 The following ASN.1 type is defined to aid the specification of the behaviour of a framework, but its abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
BSPUnloadCallParams ::= SEQUENCE {
        bspUuid                     BioAPI-UUID,
        unitEventHandlerAddress     MemoryAddress,
        unitEventHandlerContext     MemoryAddress
    }
```

16.10.4 When a framework receives a call to the function **BioAPI_BSPUnload** from the local application, it shall first determine the hosting endpoint and the product UUID of the BSP (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23, and then one of the three following subclauses applies.

16.10.4.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPUuid** shall be set by converting from *bspProductUuid* as specified in clause 19 in conjunction with 15.58;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create a temporary abstract value (say, *bspUnloadCallParams*) of type **BSPUnloadCallParams** (see 16.10.3) by converting from the parameters of the **BioAPI_BSPUnload** function call as specified in 16.10.6;

d) search the **RunningBSPLocalReferences** table (see 18.5) for an entry where:

1) the component **hostingEndpointIRI** contains the local endpoint IRI;

2) the component **bspProductUuid** has the value *bspProductUuid*;

3) if the component **bspUuid** of *bspUnloadCallParams* has a value different from *bspProductUuid*, then the component **useBSPAccessUuid** of the entry has the value **TRUE**; otherwise, it has the value **FALSE**; and

4) the components **unitEventHandlerAddress** and **unitEventHandlerContext** have the same values as the components of *bspUnloadCallParams* with the same names;

e) if such an entry is not found, then return the value **BioAPIERR_NOT_A_RUNNING_BSP** to the local application, skipping the remaining actions;

f) delete the entry of the **RunningBSPLocalReferences** table (subclause 18.5.3 applies);

NOTE – If multiple entries are found, any one of those entries (but exactly one) will be deleted.

g) return the value 0 to the local application.

16.10.4.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *outgoingRequestParams*) of type **BSPUnload-RequestParams** (see 16.10.2) where:

1) the component **bspProductUuid** shall be set to *bspProductUuid*; and

2) if the component **unitEventHandlerAddress** has a value different from 0, then the component **unitEventSubscription** of *outgoingRequestParams* shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

b) create and send a **bspUnload** request BIP message (see 13.2) with the slave endpoint IRI set to the hosting endpoint IRI and the parameter value set to *outgoingRequestParams*;

c) receive a corresponding **bspUnload** response BIP message (see 13.6);

d) if the return value of the response BIP message is not 0, then return that value to the local application, skipping the remaining actions;

e) create a temporary abstract value (say, *bspUnloadCallParams*) of type **BSPUnloadCallParams** (see 16.10.3) by converting from the parameters of the **BioAPI_BSPUnload** function call as specified in 16.10.6;

f) search the **RunningBSPLocalReferences** table (see 18.5) for an entry where:

1) the component **hostingEndpointIRI** contains the hosting endpoint IRI;

2) the component **bspProductUuid** has the value *bspProductUuid*;

3) if the component **bspUuid** of *bspUnloadCallParams* has a value different from *bspProductUuid*, then the component **useBSPAccessUuid** of the entry has the value **TRUE**; otherwise, it has the value **FALSE**; and

4) the components **unitEventHandlerAddress** and **unitEventHandlerContext** have the same values as the components of *bspUnloadCallParams* with the same names;

g) if such an entry is not found, then return the value **BioAPIERR_NOT_A_RUNNING_BSP** to the local application, skipping the remaining actions;

h) delete the entry of the **RunningBSPLocalReferences** table (subclause 18.5.3 applies);

NOTE – If multiple entries are found, any one of those entries (but exactly one) will be deleted.

i) return the value 0 to the local application.

16.10.4.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.10.5 When a framework receives (see 13.9) a **bspUnload** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **BSPUnload-RequestParams** – see 16.10.2) of the **bspUnload** request BIP message;

b) create a temporary abstract value (say, *bspUnloadCallParams*) of type **BSPUnloadCallParams** (see 16.10.3) where:

1) the component **bspUuid** shall be set from the component **bspProductUuid** of *incomingRequestParams*; and

2) the components **unitEventHandlerAddress** and **unitEventHandlerContext** shall be set to 0;

c) make an internal BioAPI function call (see 13.10) to the function **BioAPI_BSPUnload**, where the parameters of the function call shall be set by converting from *bspUnloadCallParams* as specified in 16.10.6;

d) if the return value of the internal call is not 0, then create and send a corresponding **bspUnload** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

e) search the **RunningBSPRemoteReferences** table (see 18.6) for an entry where the component **referrerEndpointIRI** contains the master endpoint IRI and the components **bspProductUuid** and **unitEventSubscription** have the same values as the components of *incomingRequestParams* with the same names;

f) if there is no such entry, then create and send a corresponding **bspUnload** response BIP message (see 13.3) with the return value set to **BioAPIERR_NOT_A_RUNNING_BSP**, skipping the remaining actions;

g) delete the entry of the **RunningBSPRemoteReferences** table (subclause 18.6.3 applies);

NOTE – If multiple entries are found, any one of those entries (but exactly one) will be deleted.

h) create and send a corresponding **bspUnload** response BIP message (see 13.3) with the parameter value set to **NULL** and the return value set to 0.

16.10.6 Conversion between the parameters of the C function **BioAPI_BSPUnload** and the ASN.1 type **BSPUnloadCallParams** (see 16.10.3) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 42.

Table 42 – Mapping between the parameters of the function BioAPI_BSPUnload and the ASN.1 type BSPUnloadCallParams

Function parameter	Component of the ASN.1 type	References
BSPUuid	bspUuid	Clause 19 in conjunction with 15.58
EventHandler	unitEventHandlerAddress	15.1.7
EventHandlerCtx	unitEventHandlerContext	15.1.7

16.11 Function BioAPI_QueryUnits

16.11.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_QueryUnits
 (const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_SCHEMA **UnitSchemaArray,
        uint32_t *NumberOfElements);
```

16.11.2 A pair of BIP message types are related to this function: the **queryUnits** request BIP message type and the **queryUnits** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
QueryUnits-RequestParams ::= SEQUENCE {
        bspProductUuid          BioAPI-UUID
    }
```

and:

```
QueryUnits-ResponseParams ::= SEQUENCE {
        unitSchemas             SEQUENCE (SIZE(0..max-unsigned-int)) OF
                                    unitSchema BioAPI-UNIT-SCHEMA
    }
```

16.11.3 When a framework receives a call to the function **BioAPI_QueryUnits** from the local application, it shall first determine the hosting endpoint and the product UUID of the BSP (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call,

except that the parameter **BSPUuid** shall be set by converting from *bspProductUuid* as specified in clause 19 in conjunction with 15.58, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **queryUnits** request/response BIP message pair as specified in clause 27, using 16.11.5 and 16.11.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.11.4 When a framework receives (see 13.9) a **queryUnits** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_QueryUnits** to create and send a corresponding **queryUnits** response BIP message as specified in clause 28, using 16.11.5 and 16.11.6 to convert between function parameters and ASN.1 components when required within that clause.

16.11.5 Conversion between the parameters of the C function **BioAPI_QueryUnits** and the ASN.1 type **QueryUnits-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 43.

Table 43 – Mapping between the parameters of the function BioAPI_QueryUnits and the ASN.1 type QueryUnits-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPUuid	bspProductUuid	clause 25
UnitSchemaArray	*none*	clause 22
NumberOfElements	*none*	clause 22

16.11.6 Conversion between the parameters of the C function **BioAPI_QueryUnits** and the ASN.1 type **QueryUnits-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 44.

Table 44 – Mapping between the parameters of the function BioAPI_QueryUnits and the ASN.1 type QueryUnits-ResponseParams

Function parameter	Component of the ASN.1 type	References
UnitSchemaArray, NumberOfElements	unitSchemas	clause 20 in conjunction with 16.11.7 and 16.11.8

16.11.7 Conversion from the pair of C variables pointed to by the parameters **UnitSchemaArray**/**NumberOfElements** to the ASN.1 component **unitSchemas** shall be done as follows: Calling *N* the value of the C variable pointed to by the parameter **NumberOfElements**, each of the first *N* elements (of type **BioAPI_UNIT_SCHEMA** – see 15.57) in the array pointed to by the C variable pointed to by the parameter **UnitSchemaArray** shall be converted, in order, to an element of the component **unitSchemas** as specified in 15.57. The component **unitSchemas** shall have exactly *N* elements.

16.11.8 Conversion from the ASN.1 component **unitSchemas** to the pair of C variables pointed to by the parameters **UnitSchemaArray**/**NumberOfElements** shall be done as follows: Calling *N* the number of elements of the component **unitSchemas**, a newly allocated array of *N* elements of type **BioAPI_UNIT_SCHEMA** (see 15.57) shall be filled by converting each element of the component **unitSchemas**, in order, to an element of the array as specified in 15.57. The C variable pointed to by the parameter **UnitSchemaArray** shall be set to the address of the array, and the C variable pointed to by the parameter **NumberOfElements** shall be set to *N*.

16.12 Function BioAPI_QueryBFPs

16.12.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_QueryBFPs
        (const BioAPI_UUID *BSPUuid,
        BioAPI_BFP_LIST_ELEMENT **BFPList,
        uint32_t *NumberOfElements);
```

16.12.2 A pair of BIP message types are related to this function: the **queryBFPs** request BIP message type and the **queryBFPs** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
QueryBFPs-RequestParams ::= SEQUENCE {
        bspProductUuid          BioAPI-UUID
}
```

and:

```
QueryBFPs-ResponseParams ::= SEQUENCE {
        bfps                    SEQUENCE (SIZE(0..max-unsigned-int)) OF
                                    bfp BioAPI-BFP-LIST-ELEMENT
}
```

16.12.3 When a framework receives a call to the function **BioAPI_QueryBFPs** from the local application, it shall first determine the hosting endpoint and the product UUID of the BSP (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values, except that the parameter **BSPUuid** shall be set by converting from *bspProductUuid* as specified in clause 19 in conjunction with 15.58, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **queryBFPs** request/response BIP message pair as specified in clause 27, using 16.12.5 and 16.12.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.12.4 When a framework receives (see 13.9) a **queryBFPs** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_QueryBFPs** to create and send a corresponding **queryBFPs** response BIP message as specified in clause 28, using 16.12.5 and 16.12.6 to convert between function parameters and ASN.1 components when required within that clause.

16.12.5 Conversion between the parameters of the C function **BioAPI_QueryBFPs** and the ASN.1 type **QueryBFPs-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 45.

Table 45 – Mapping between the parameters of the function BioAPI_QueryBFPs and the ASN.1 type QueryBFPs-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPUuid	bspProductUuid	clause 25
BFPList	*None*	clause 22
NumberOfElements	*None*	clause 22

16.12.6 Conversion between the parameters of the C function **BioAPI_QueryBFPs** and the ASN.1 type **QueryBFPs-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 46.

Table 46 – Mapping between the parameters of the function BioAPI_QueryBFPs and the ASN.1 type QueryBFPs-ResponseParams

Function parameter	Component of the ASN.1 type	References
BFPList, NumberOfElements	bfps	clause 20 in conjunction with 16.12.7 and 16.12.8

16.12.7 Conversion from the pair of C variables pointed to by the parameters **BFPList**/**NumberOfElements** to the ASN.1 component **bfps** shall be done as follows: Calling *N* the value of the C variable pointed to by the parameter **NumberOfElements**, each of the first *N* elements (of type **BioAPI_BFP_LIST_ELEMENT** – see 15.4) in the array pointed to by the C variable pointed to by the parameter **BFPList** shall be converted, in order, to an element of the component **bfps** as specified in 15.4. The component **bfps** shall have exactly *N* elements.

16.12.8 Conversion from the ASN.1 component **bfps** to the pair of C variables pointed to by the parameters **BFPList**/**NumberOfElements** shall be done as follows: Calling *N* the number of elements of the component **bfps**, a newly allocated array of *N* elements of type **BioAPI_BFP_LIST_ELEMENT** (see 15.4) shall be filled by converting each element of the component **bfps**, in order, to an element of the array as specified in 15.4. The C variable pointed to

by the parameter **BFPList** shall be set to the address of the array, and the C variable pointed to by the parameter **NumberOfElements** shall be set to *N*.

16.13 Function BioAPI_BSPAttach

16.13.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_BSPAttach
        (const BioAPI_UUID *BSPUuid,
        BioAPI_VERSION Version,
        const BioAPI_UNIT_LIST_ELEMENT *UnitList,
        uint32_t NumUnits,
        BioAPI_HANDLE *NewBSPHandle);
```

16.13.2 A pair of BIP message types are related to this function: the **bspAttach** request BIP message type and the **bspAttach** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
BSPAttach-RequestParams ::= SEQUENCE {
        bspProductUuid          BioAPI-UUID,
        version                 BioAPI-VERSION,
        units                   SEQUENCE
                                    (SIZE(0..max-unsigned-int)) OF
                                        unit BioAPI-UNIT-LIST-ELEMENT
    }
```

and:

```
BSPAttach-ResponseParams ::= SEQUENCE {
        newOriginalBSPHandle    BioAPI-HANDLE
    }
```

16.13.3 The following ASN.1 type is defined to aid the specification of the behaviour of a framework, but its abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
BSPAttachCallOutputParams ::= SEQUENCE {
        newBSPHandle            BioAPI-HANDLE
    }
```

16.13.4 When a framework receives a call to the function **BioAPI_BSPAttach** from the local application, it shall first determine the hosting endpoint and the product UUID of the BSP (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23, and then one of the three following subclauses applies.

16.13.4.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPUuid** shall be set by converting from *bspProductUuid* as specified in clause 19 in conjunction with 15.58;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create a temporary abstract value (say, *incomingResponseParams*) of type **BSPAttach-ResponseParams** (see 16.13.2) by converting from the output parameters of the internal call as specified in 16.13.9;

d) create a temporary abstract value (say, *incomingRequestParams*) of type **BSPAttach-RequestParams** (see 16.13.2) by converting from the parameters of the **BioAPI_BSPAttach** function call as specified 16.13.6;

e) create a temporary abstract value (say, *localBSPHandle*) of type **BioAPI-HANDLE** (see 15.42), which may be any value of that ASN.1 type different from the value of the component **localBSPHandle** in all current entries of the **AttachSessionLocalReferences** table (see 18.8);

NOTE – The method for generating this abstract value is not specified in this Recommendation | International Standard.

f) add an entry to the **AttachSessionLocalReferences** table (see 18.8), where:

1) the component **hostingEndpointIRI** shall be set to the hosting endpoint IRI;

2) the component **bspProductUuid** shall be set to *bspProductUuid*;

3) if the component **bspUuid** of *incomingRequestParams* has a value different from *bspProductUuid*, then the component **useBSPAccessUuid** of the entry shall be set to **TRUE**; otherwise, it shall be set to **FALSE**; and

4) the component **originalBSPHandle** shall be set from the component **newOriginalBSPHandle** of *incomingResponseParams*; and

5) the component **localBSPHandle** shall be set to *localBSPHandle*;

g) create a temporary abstract value (say, *outgoingResponseParams*) of type **BSPAttachCallOutputParams** (see 16.13.3), where the component **newBSPHandle** shall be set to *localBSPHandle*;

h) set the output parameters of the **BioAPI_BSPAttach** function call by converting from *outgoingResponseParams* as specified in 16.13.10;

i) return the value 0 to the local application.

16.13.4.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *incomingRequestParams*) of type **BSPAttach-RequestParams** (see 16.13.2) by converting from the parameters of the **BioAPI_BSPAttach** function call as specified in 16.13.6;

b) create and send a **bspAttach** request BIP message (see 13.2) with the slave endpoint IRI set to the hosting endpoint IRI and the parameter value set to *incomingRequestParams*;

c) receive a corresponding **bspAttach** response BIP message (see 13.6);

d) if the return value of the response BIP message is not 0, then return that value to the local application, skipping the remaining actions;

e) let *incomingResponseParams* be the parameter value (of type **BSPAttach-ResponseParams** – see 16.13.2) of the **bspAttach** response BIP message;

f) create a temporary abstract value (say, *localBSPHandle*) of type **BioAPI-HANDLE** (see 15.42), which may be any value of that ASN.1 type different from the value of the component **localBSPHandle** in all current entries of the **AttachSessionLocalReferences** table (see 18.8);

NOTE – The method for generating this abstract value is not specified in this Recommendation | International Standard.

g) add an entry to the **AttachSessionLocalReferences** table (see 18.8), where:

1) the component **hostingEndpointIRI** shall be set to the hosting endpoint IRI;

2) the component **bspProductUuid** shall be set to *bspProductUuid*;

3) if the component **bspUuid** of *incomingRequestParams* has a value different from *bspProductUuid*, then the component **useBSPAccessUuid** of the entry shall be set to **TRUE**; otherwise, it shall be set to **FALSE**; and

4) the component **originalBSPHandle** shall be set from the component **newOriginalBSPHandle** of *incomingResponseParams*; and

5) the component **localBSPHandle** shall be set to *localBSPHandle*;

h) create a temporary abstract value (say, *outgoingResponseParams*) of type **BSPAttachCallOutputParams** (see 16.13.3), where the component **newBSPHandle** shall be set to *localBSPHandle*;

i) set the output parameters of the **BioAPI_BSPAttach** function call by converting from *outgoingResponseParams* as specified in 16.13.10;

j) return the value 0 to the local application.

16.13.4.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.13.5 When a framework receives (see 13.9) a **bspAttach** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **BSPAttach-RequestParams** – see 16.13.2) of the **bspAttach** request BIP message;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_BSPAttach** where the parameters of the function call shall be set by converting from *incomingRequestParams* as specified in 16.13.6;

c) if the return value of the internal call is not 0, then create and send a corresponding **bspAttach** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

d) create a temporary abstract value (say, *incomingResponseParams*) of type **BSPAttach-ResponseParams** (see 16.13.2) by converting from the output parameters of the internal call as specified in 16.13.9;

e) add an entry to the **AttachSessionRemoteReferences** table (see 18.9), where:

1) the component **referrerEndpointIRI** shall be set to the master endpoint IRI;

2) the component **bspProductUuid** shall be set from the component **bspProductUuid** of *incomingRequestParams*; and

3) the component **originalBSPHandle** shall be set from the component **newOriginalBSPHandle** of *incomingResponseParams*;

f) create and send a corresponding **bspAttach** response BIP message (see 13.3) with the parameter value set to *incomingResponseParams* and the return value set to 0.

16.13.6 Conversion between the parameters of the C function **BioAPI_BSPAttach** and the ASN.1 type **BSPAttach-RequestParams** (see 16.13.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 47.

Table 47 – Mapping between the parameters of the function BioAPI_BSPAttach and the ASN.1 type BSPAttach-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPUuid	bspProductUuid	clause 25
Version	version	15.59
UnitList, NumUnits	units	16.13.7
NewBSPHandle	*none*	clause 22

16.13.7 Conversion from the pair of C parameters **UnitList**/**NumUnits** to the ASN.1 component **units** shall be done as follows: Calling *N* the value of the parameter **NumUnits**, each of the first *N* elements (of type **BioAPI_UNIT_LIST_ELEMENT** – see 15.56) in the array pointed to by the parameter **UnitList** shall be converted, in order, to an element of the component **units** as specified in 15.56. The component **units** shall have exactly *N* elements.

16.13.8 Conversion from the ASN.1 component **units** to the pair of C parameters **UnitList**/**NumUnits** shall be done as follows: Calling *N* the number of elements of the component **units**, a newly allocated array of *N* elements of type **BioAPI_UNIT_LIST_ELEMENT** (see 15.56) shall be filled by converting each element of the component **units**, in order, to an element of the array as specified in 15.56. The C parameter **UnitList** shall be set to the address of the array, and the C parameter **NumUnits** shall be set to *N*.

16.13.9 Conversion from the parameters of the C function **BioAPI_BSPAttach** to the ASN.1 type **BSPAttach-ResponseParams** (see 16.13.2) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 48.

Table 48 – Mapping from the parameters of the function BioAPI_BSPAttach to the ASN.1 type BSPAttach-ResponseParams

Function parameter	Component of the ASN.1 type	References
NewBSPHandle	newOriginalBSPHandle	clause 29 in conjunction with 15.42

16.13.10 Conversion from the ASN.1 type **BSPAttach-ResponseParams** (see 16.13.2) to the parameters of the C function **BioAPI_BSPAttach** shall be done by converting from individual ASN.1 components to function parameters in accordance with Table 49.

Table 49 – Mapping from the ASN.1 type BSPAttach-ResponseParams to the parameters of the function BioAPI_BSPAttach

Component of the ASN.1 type	Function parameter	References
newBSPHandle	NewBSPHandle	clause 20 in conjunction with 15.42

16.14 Function BioAPI_BSPDetach

16.14.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_BSPDetach
        (BioAPI_HANDLE BSPHandle);
```

16.14.2 A pair of BIP message types are related to this function: the **bspDetach** request BIP message type and the **bspDetach** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
BSPDetach-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE
    }
```

and:

```
BSPDetach-ResponseParams ::= NULL
```

16.14.3 When a framework receives a call to the function **BioAPI_BSPDetach** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24, and then one of the three following subclauses applies.

16.14.3.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) search the **AttachSessionLocalReferences** table (see 18.8) for an entry where the value of the component **originalBSPHandle** is *originalBSPHandle*;

d) if there is no such entry, then return the value **BioAPIERR_NOT_A_RUNNING_BSP** to the local application, skipping the remaining actions;

e) delete the entry of the **AttachSessionLocalReferences** table (subclause 18.8.3 applies);

f) return the value 0 to the local application.

16.14.3.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *incomingRequestParams*) of type **BSPDetach-RequestParams** (see 16.14.2) by converting from the parameters of the **BioAPI_BSPDetach** function call as specified in 16.14.5;

b) create and send a **bspDetach** request BIP message (see 13.2) with the slave endpoint IRI set to the hosting endpoint IRI and the parameter value set to *incomingRequestParams*;

c) receive a corresponding **bspDetach** response BIP message (see 13.6);

d) if the return value of the response BIP message is not 0, then return that value to the local application, skipping the remaining actions;

e) search the **AttachSessionLocalReferences** table (see 18.8) for an entry where the value of the component **originalBSPHandle** is *originalBSPHandle*;

f) if there is no such entry, then return the value **BioAPIERR_NOT_A_RUNNING_BSP** to the local application, skipping the remaining actions;

g) delete the entry of the **AttachSessionLocalReferences** table (subclause 18.8.3 applies);

h) return the value 0 to the local application.

16.14.3.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.14.4 When a framework receives (see 13.9) a **bspDetach** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **BSPDetach-RequestParams** – see 16.14.2) of the **bspDetach** request BIP message;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_BSPDetach**, where the parameters of the function call shall be set by converting from *incomingRequestParams* as specified in 16.14.5;

c) if the return value of the internal call is not 0, then create and send a corresponding **bspDetach** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

d) search the **AttachSessionRemoteReferences** table (see 18.9) for an entry where the component **originalBSPHandle** has the same value as the component **originalBSPHandle** of *incomingRequestParams*;

e) if there is no such entry, then create and send a corresponding **bspDetach** response BIP message (see 13.3) with the return value set to **BioAPIERR_NOT_A_RUNNING_BSP**, skipping the remaining actions;

f) delete the entry of the **AttachSessionRemoteReferences** table (subclause 18.9.3 applies);

g) create and send a corresponding **bspDetach** response BIP message (see 13.3) with the parameter value set to **NULL** and the return value set to 0.

16.14.5 Conversion between the parameters of the C function **BioAPI_BSPDetach** and the ASN.1 type **BSPDetach-RequestParams** (see 16.14.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 50.

Table 50 – Mapping between the parameters of the function BioAPI_BSPDetach and the ASN.1 type BSPDetach-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26

16.15 Function BioAPI_EnableEvents

16.15.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_EnableEvents
        (BioAPI_HANDLE BSPHandle,
        BioAPI_EVENT_MASK Events);
```

16.15.2 A pair of BIP message types are related to this function: the **enableEvents** request BIP message type and the **enableEvents** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
EnableEvents-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        unitEvents              BioAPI-UNIT-EVENT-TYPE-MASK
    }
```

and:

```
EnableEvents-ResponseParams ::= NULL
```

16.15.3 When a framework receives a call to the function **BioAPI_EnableEvents** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint an **enableUnitEvents** request/response BIP message pair as specified in clause 27, using 16.15.5 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.15.4 When a framework receives (see 13.9) an **enableUnitEvents** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_EnableEvents** to create and send a corresponding **enableUnitEvents** response BIP message as specified in clause 28, using 16.19.5 to convert between function parameters and ASN.1 components when required within that clause.

16.15.5 Conversion between the parameters of the C function **BioAPI_EnableEvents** and the ASN.1 type **EnableUnitEvents-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 51.

Table 51 – Mapping between the parameters of the function BioAPI_EnableEvents and the ASN.1 type EnableUnitEvents-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
Events	unitEvents	15.31

16.16 Function BioAPI_EnableEventNotifications

16.16.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_EnableEventNotifications
        (const BioAPI_UUID *BSPUuid,
        BioAPI_EVENT_MASK Events);
```

16.16.2 A pair of BIP message types are related to this function: the **EnableEventNotifications** request BIP message type and the **EnableEventNotifications** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
EnableEventNotifications-RequestParams ::= SEQUENCE {
        bspProductUuid          BioAPI-UUID,
        unitEventTypes          BioAPI-UNIT-EVENT-TYPE-MASK
    }
```

and:

```
EnableEventNotifications-ResponseParams ::= NULL
```

16.16.3 The following ASN.1 type is defined to aid the specification of the behaviour of a framework, but its abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
EnableCallParams ::= SEQUENCE {
        bspUuid                 BioAPI-UUID,
        unitEventTypes          BioAPI-UNIT-EVENT-TYPE-MASK
    }
```

16.16.4 When a framework receives a call to the function **BioAPI_EnableEventNotifications** from the local application, it shall first determine the hosting endpoint and the product UUID of the BSP (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23, and then one of the three following subclauses applies.

16.16.4.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *enableCallParams*) of type **EnableCallParams** (see 16.16.3) by converting from the parameters of the **BioAPI_EnableEventNotifications** function call as specified in 16.16.6;

b) if the **UnitEventNotificationDisablers** table (see 18.7) has an entry where the component **referrerEndpointIRI** is set to the local endpoint IRI and the component **bspProductUuid** is set to *bspProductUuid*, then delete that entry;

c) if the component **unitEventTypes** of *enableCallParams* indicates that at least one type of event notification is to be disabled, then add an entry to the **UnitEventNotificationDisablers** table (see 18.7), where:

   1) the component **referrerEndpointIRI** shall be set to the local endpoint IRI;

   2) the component **bspProductUuid** shall be set to *bspProductUuid*; and

   3) the component **unitEventTypes** shall be set to the component **unitEventTypes** of *enableCallParams*;

d) return the value 0 to the local application.

16.16.4.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *enableCallParams*) of type **EnableCallParams** (see 16.16.3) by converting from the parameters of the **BioAPI_EnableEventNotifications** function call as specified in 16.16.6;

b) create a temporary abstract value (say, *outgoingRequestParams*) of type **EnableEventNotifications-RequestParams** (see 16.16.2) where:

1) the component **bspProductUuid** shall be set to *bspProductUuid*; and

2) the component **unitEventTypes** shall be set to the component **unitEventTypes** of *enableCallParams*;

c) create and send an **enableEventNotifications** request BIP message (see 13.2) with the slave endpoint IRI set to the hosting endpoint IRI and the parameter value set to *outgoingRequestParams*;

d) receive a corresponding **enableEventNotifications** response BIP message (see 13.6);

e) if the return value of the response BIP message is not 0, then delete the entry added above to the **UnitEventNotificationDisablers** table, and return that value to the local application, skipping the remaining actions;

f) return 0 to the local application.

16.16.4.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.16.5 When a framework receives (see 13.9) an **enableEventNotifications** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **EnableEventNotifications-RequestParams** – see 16.16.2) of the **enableEventNotifications** request BIP message;

b) if the **UnitEventNotificationDisablers** table (see 18.7) has an entry where the component **referrerEndpointIRI** is set to the master endpoint IRI and the component **bspProductUuid** has the same value as the component **bspProductUuid** of *incomingRequestParams*, then delete that entry;

c) if the component **unitEventTypes** of *incomingRequestParams* indicates that at least one type of event notification is to be disabled, then add an entry to the **UnitEventNotificationDisablers** table (see 18.7), where:

1) the component **referrerEndpointIRI** shall be set to the master endpoint IRI;

2) the component **bspProductUuid** shall be set from the component **bspProductUuid** of *incomingRequestParams*;

3) the component **unitEventTypes** shall be set from the component **unitEventTypes** of *incomingRequestParams*;

d) create and send a corresponding **enableEventNotifications** response BIP message (see 13.3) with the return value set to 0.

16.16.6 Conversion between the parameters of the C function **BioAPI_EnableEventNotifications** and the ASN.1 type **EnableCallParams** (see 16.16.3) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 52.

Table 52 – Mapping between the parameters of the function BioAPI_EnableEventNotifications and the ASN.1 type EnableCallParams

Function parameter	Component of the ASN.1 type	References
BSPUuid	bspUuid	clause 19 in conjunction with 15.58
Events	unitEventTypes	15.31

16.17 Function BioAPI_ControlUnit

16.17.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_ControlUnit
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_UNIT_ID UnitID,
        uint32_t ControlCode,
        const BioAPI_DATA *InputData,
        BioAPI_DATA *OutputData);
```

16.17.2 A pair of BIP message types are related to this function: the **controlUnit** request BIP message type and the **controlUnit** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
ControlUnit-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        unitID                  BioAPI-UNIT-ID,
        controlCode             UnsignedInt,
        inputData               BioAPI-DATA
    }
```

and:

```
ControlUnit-ResponseParams ::= SEQUENCE {
        outputData              BioAPI-DATA
    }
```

16.17.3 When a framework receives a call to the function **BioAPI_ControlUnit** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **controlUnit** request/response BIP message pair as specified in clause 27, using 16.17.5 and 16.17.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.17.4 When a framework receives (see 13.9) a **controlUnit** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_ControlUnit** to create and send a corresponding **controlUnit** response BIP message as specified in clause 28, using 16.17.5 and 16.17.6 to convert between function parameters and ASN.1 components when required within that clause.

16.17.5 Conversion between the parameters of the C function **BioAPI_ControlUnit** and the ASN.1 type **ControlUnit-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 53.

Table 53 – Mapping between the parameters of the function BioAPI_ControlUnit and the ASN.1 type ControlUnit-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
UnitID	unitID	15.55
ControlCode	controlCode	15.1.5
InputData	inputData	clause 19 in conjunction with 15.22
OutputData	*none*	clause 22

16.17.6 Conversion between the parameters of the C function **BioAPI_ControlUnit** and the ASN.1 type **ControlUnit-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 54.

Table 54 – Mapping between the parameters of the function BioAPI_ControlUnit and the ASN.1 type ControlUnit-ResponseParams

Function parameter	Component of the ASN.1 type	References
OutputData	outputData	clause 20 in conjunction with 15.22

16.18 Function BioAPI_Control

16.18.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Control
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_UNIT_ID UnitID,
        const BioAPI_UUID *ControlCode,
        const BioAPI_DATA *InputData,
        BioAPI_DATA *OutputData);
```

16.18.2 A pair of BIP message types are related to this function: the **control** request BIP message type and the **control** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
Control-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        unitID                  BioAPI-UNIT-ID,
        controlCode             BioAPI-UUID,
        inputData               BioAPI-DATA
    }
```

and:

```
Control-ResponseParams ::= SEQUENCE {
        outputData              BioAPI-DATA
    }
```

16.18.3 When a framework receives a call to the function **BioAPI_Control** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a control request/response BIP message pair as specified in clause 27, using 16.18.5 and 16.18.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.18.4 When a framework receives (see 13.9) a control request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_Control** to create and send a corresponding control response BIP message as specified in 13.10, using 16.18.5 and 16.18.6 and to convert between function parameters and ASN.1 components when required within that clause.

16.18.5 Conversion between the parameters of the C function **BioAPI_Control** and the ASN.1 type **Control-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 55.

Table 55 – Mapping between the parameters of the function BioAPI_Control and the ASN.1 type Control-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
UnitID	unitID	15.55
ControlCode	controlCode	15.1.5
InputData	inputData	clause 19 in conjunction with 15.22
OutputData	*None*	clause 22

16.18.6 Conversion between the parameters of the C function **BioAPI_Control** and the ASN.1 type **Control-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 56.

Table 56 – Mapping between the parameters of the function BioAPI_Control and the ASN.1 type Control-ResponseParams

Function parameter	Component of the ASN.1 type	References
OutputData	outputData	clause 20 in conjunction with 15.22

16.19 Function BioAPI_FreeBIRHandle

16.19.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_FreeBIRHandle
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_BIR_HANDLE  Handle);
```

16.19.2 A pair of BIP message types are related to this function: the **freeBIRHandle** request BIP message type and the **freeBIRHandle** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
FreeBIRHandle-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        birHandle               BioAPI-BIR-HANDLE
    }
```

and:

```
FreeBIRHandle-ResponseParams ::= NULL
```

16.19.3 When a framework receives a call to the function **BioAPI_FreeBIRHandle** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **freeBIRHandle** request/response BIP message pair as specified in clause 27, using 16.19.5 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.19.4 When a framework receives (see 13.9) a **freeBIRHandle** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_FreeBIRHandle** to create and send a corresponding **freeBIRHandle** response BIP message as specified in clause 28, using 16.19.5 to convert between function parameters and ASN.1 components when required within that clause.

16.19.5 Conversion between the parameters of the C function **BioAPI_FreeBIRHandle** and the ASN.1 type **FreeBIRHandle-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 57.

Table 57 – Mapping between the parameters of the function BioAPI_FreeBIRHandle and the ASN.1 type FreeBIRHandle-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
Handle	birHandle	15.12

16.20 Function BioAPI_GetBIRFromHandle

16.20.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_GetBIRFromHandle
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_BIR_HANDLE  Handle,
        BioAPI_BIR *BIR);
```

16.20.2 A pair of BIP message types are related to this function: the **getBIRFromHandle** request BIP message type and the **getBIRFromHandle** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
GetBIRFromHandle-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        birHandle               BioAPI-BIR-HANDLE
    }
```

and:

```
GetBIRFromHandle-ResponseParams ::= SEQUENCE {
        bir                     BioAPI-BIR
    }
```

16.20.3 When a framework receives a call to the function **BioAPI_GetBIRFromHandle** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **getBIRFromHandle** request/response BIP message pair as specified in clause 27, using 16.20.5 and 16.20.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.20.4 When a framework receives (see 13.9) a **getBIRFromHandle** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_GetBIRFromHandle** to create and send a corresponding **getBIRFromHandle** response BIP message as specified in clause 28, using 16.20.5 and 16.20.6 to convert between function parameters and ASN.1 components when required within that clause.

16.20.5 Conversion between the parameters of the C function **BioAPI_GetBIRFromHandle** and the ASN.1 type **GetBIRFromHandle-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 58.

Table 58 – Mapping between the parameters of the function BioAPI_GetBIRFromHandle and the ASN.1 type GetBIRFromHandle-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
Handle	birHandle	15.12
BIR	*none*	clause 22

16.20.6 Conversion between the parameters of the C function **BioAPI_GetBIRFromHandle** and the ASN.1 type **GetBIRFromHandle-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 59.

Table 59 – Mapping between the parameters of the function BioAPI_GetBIRFromHandle and the ASN.1 type GetBIRFromHandle-ResponseParams

Function parameter	Component of the ASN.1 type	References
BIR	bir	clause 20 in conjunction with 15.6

16.21 Function BioAPI_GetHeaderFromHandle

16.21.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_GetHeaderFromHandle
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_BIR_HANDLE Handle,
        BioAPI_BIR_HEADER *Header);
```

16.21.2 A pair of BIP message types are related to this function: the **getHeaderFromHandle** request BIP message type and the **getHeaderFromHandle** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
GetHeaderFromHandle-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        birHandle               BioAPI-BIR-HANDLE
    }
```

and:

```
GetHeaderFromHandle-ResponseParams ::= SEQUENCE {
        header                  BioAPI-BIR-HEADER
    }
```

16.21.3 When a framework receives a call to the function **BioAPI_GetHeaderFromHandle** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **getHeaderFromHandle** request/response BIP message pair as specified in clause 27, using 16.21.5 and 16.21.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.21.4 When a framework receives (see 13.9) a **getHeaderFromHandle** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_GetHeaderFromHandle** to create and send a corresponding **getHeaderFromHandle** response BIP message as specified in clause 28, using 16.21.5 and 16.21.6 to convert between function parameters and ASN.1 components when required within that clause.

16.21.5 Conversion between the parameters of the C function **BioAPI_GetHeaderFromHandle** and the ASN.1 type **GetHeaderFromHandle-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 60.

Table 60 – Mapping between the parameters of the function BioAPI_GetHeaderFromHandle and the ASN.1 type GetHeaderFromHandle-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
Handle	birHandle	15.12
Header	*none*	clause 22

16.21.6 Conversion between the parameters of the C function **BioAPI_GetHeaderFromHandle** and the ASN.1 type **GetHeaderFromHandle-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 61.

Table 61 – Mapping between the parameters of the function BioAPI_GetHeaderFromHandle and the ASN.1 type GetHeaderFromHandle-ResponseParams

Function parameter	Component of the ASN.1 type	References
Header	header	clause 20 in conjunction with 15.13

16.22 Function BioAPI_SubscribeToGUIEvents

16.22.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_SubscribeToGUIEvents
        (const BioAPI_UUID *GUIEventSubscriptionUuid,
        const BioAPI_UUID *BSPUuid,
        const BioAPI_HANDLE *BSPHandle,
        BioAPI_GUI_SELECT_EVENT_HANDLER GUISelectEventHandler,
        void *GUISelectEventHandlerCtx,
        BioAPI_GUI_STATE_EVENT_HANDLER GUIStateEventHandler,
        void *GUIStateEventHandlerCtx,
        BioAPI_GUI_PROGRESS_EVENT_HANDLER GUIProgressEventHandler,
        void *GUIProgressEventHandlerCtx);
```

16.22.2 A pair of BIP message types are related to this function: the **subscribeToGUIEvents** request BIP message type and the **subscribeToGUIEvents** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
SubscribeToGUIEvents-RequestParams ::= SEQUENCE {
        guiEventSubscriptionUuid        BioAPI-UUID OPTIONAL,
        bspProductUuid                  BioAPI-UUID OPTIONAL,
        originalBSPHandle               BioAPI-HANDLE OPTIONAL,
        guiSelectEventSubscribed        BOOLEAN,
        guiStateEventSubscribed         BOOLEAN,
        guiProgressEventSubscribed      BOOLEAN
}
```

and:

```
SubscribeToGUIEvents-ResponseParams ::= NULL
```

16.22.3 The following ASN.1 type is defined to aid the specification of the behaviour of a framework, but its abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
SubscribeToGUIEventsCallParams ::= SEQUENCE {
        guiEventSubscriptionUuid        BioAPI-UUID OPTIONAL,
        bspUuid                         BioAPI-UUID OPTIONAL,
        bspHandle                       BioAPI-HANDLE OPTIONAL,
        guiSelectEventHandlerAddress    MemoryAddress,
        guiSelectEventHandlerContext    MemoryAddress,
        guiStateEventHandlerAddress     MemoryAddress,
        guiStateEventHandlerContext     MemoryAddress,
        guiProgressEventHandlerAddress  MemoryAddress,
        guiProgressEventHandlerContext  MemoryAddress
}
```

16.22.4 When a framework receives a call to the function **BioAPI_SubscribeToGUIEvents** from the local application, it shall first determine either:

a) the hosting endpoint and the BSP product UUID (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23 (if the parameter **BSPUuid** has a value different from **NULL**); or

b) the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the C variable pointed to by the parameter **BSPHandle** as specified in clause 24 (if the parameter **BSPHandle** has a value different from **NULL**);

and then one of the three following subclauses applies.

NOTE – Exactly one of these two parameters is required to have the value **NULL** (see ISO/IEC 19784-1, 8.3.8.1).

16.22.4.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that:

1) if the parameter **BSPUuid** of the incoming call had a value different from **NULL**, then the parameter **BSPUuid** shall be set by converting from *bspProductUuid* as specified in clause 19 in conjunction with 15.58; and

2) if the parameter **BSPHandle** of the incoming call had a value different from **NULL**, then the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in clause 19 in conjunction with 15.42;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create a temporary abstract value (say, *subscribeToGUIEventsCallParams*) of type **SubscribeToGUIEventsCallParams** (see 16.22.3) by converting from the parameters of the **BioAPI_SubscribeToGUIEvents** function call as specified in 16.22.6;

d) add an entry to the **GUIEventLocalSubscriptions** table (see 18.10), where:

1) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of *subscribeToGUIEventsCallParams* (presence and value);

2) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

3) if the optional component **bspUuid** of *subscribeToGUIEventsCallParams* is present, then the component **bspProductUuid** of the entry shall be set to *bspProductUuid*; otherwise, it shall be set from the component **bspProductUuid** of the entry of the **AttachSessionLocalReferences** table (see 18.8) where the component **originalBSPHandle** has the value *originalBSPHandle*;

4) if the optional component **bspUuid** of *subscribeToGUIEventsCallParams* is absent or has a value different from *bspProductUuid*, then the component **useBSPAccessUuid** of the entry shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

5) if the optional component **bspHandle** of *subscribeToGUIEventsCallParams* is present, then the optional component **originalBSPHandle** of the entry shall be set to *originalBSPHandle*; otherwise, it shall be absent; and

6) the components **guiSelectEventHandlerAddress**, **guiSelectEventHandlerContext**, **guiStateEventHandlerAddress**, **guiStateEventHandlerContext**, **guiProgressEventHandlerAddress**, and **guiProgressEventHandlerContext** shall be set from the components of *subscribeToGUIEventsCallParams* with the same names;

e) return the value 0 to the local application.

16.22.4.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *subscribeToGUIEventsCallParams*) of type **SubscribeToGUIEventsCallParams** (see 16.22.3) by converting from the parameters of the **BioAPI_SubscribeToGUIEvents** function call as specified in 16.22.6;

b) create a temporary abstract value (say, *outgoingRequestParams*) of type **SubscribeToGUIEvents-RequestParams** (see 16.22.2) where:

1) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of *subscribeToGUIEventsCallParams* (presence and value);

2) if the optional component **bspUuid** of *subscribeToGUIEventsCallParams* is present, then the component **bspProductUuid** of *outgoingRequestParams* shall be set to *bspProductUuid*; otherwise, it shall be absent;

3) if the optional component **bspHandle** of *subscribeToGUIEventsCallParams* is present, then the optional component **originalBSPHandle** of *outgoingRequestParams* shall be present and shall be set to *originalBSPHandle*; otherwise, it shall be absent;

4) if the component **guiSelectEventHandlerAddress** of *subscribeToGUIEventsCallParams* has a value different from 0, then the component **guiSelectEventSubscribed** of *outgoingRequestParams* shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

5) if the component **guiStateEventHandlerAddress** of *subscribeToGUIEventsCallParams* has a value different from 0, then the component **guiStateEventSubscribed** of *outgoingRequestParams* shall be set to **TRUE**; otherwise, it shall be set to **FALSE**; and

6) if the component **guiProgressEventHandlerAddress** of *subscribeToGUIEventsCallParams* has a value different from 0, then the component **guiProgressEventSubscribed** of *outgoingRequestParams* shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

c) create and send a **subscribeToGUIEvents** request BIP message (see 13.2) with the slave endpoint IRI set to the hosting endpoint IRI and the parameter value set to *outgoingRequestParams*;

d) receive a corresponding **subscribeToGUIEvents** response BIP message (see 13.6);

e) if the return value of the response BIP message is not 0, then return that value to the local application, skipping the remaining actions;

f) add an entry to the **GUIEventLocalSubscriptions** table (see 18.10), where:

1) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of *subscribeToGUIEventsCallParams* (presence and value);

2) the component **hostingEndpointIRI** shall be set to the hosting endpoint IRI;

3) if the optional component **bspUuid** of *subscribeToGUIEventsCallParams* is present, then the component **bspProductUuid** of the entry shall be set to *bspProductUuid*; otherwise, it shall be set from the component **bspProductUuid** of the entry of the **AttachSessionLocalReferences** table (see 18.8) where the component **originalBSPHandle** has the value *originalBSPHandle*;

4) if the optional component **bspUuid** of *subscribeToGUIEventsCallParams* is absent or has a value different from *bspProductUuid*, then the component **useBSPAccessUuid** of the entry shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

5) if the optional component **bspHandle** of *subscribeToGUIEventsCallParams* is present, then the optional component **originalBSPHandle** of the entry shall be set to *originalBSPHandle*; otherwise, it shall be absent; and

6) the components **guiSelectEventHandlerAddress**, **guiSelectEventHandlerContext**, **guiStateEventHandlerAddress**, **guiStateEventHandlerContext**, **guiProgressEventHandlerAddress**, and **guiProgressEventHandlerContext** shall be set from the components of *subscribeToGUIEventsCallParams* with the same names;

g) return the value 0 to the local application.

16.22.4.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.22.5 When a framework receives (see 13.9) a **subscribeToGUIEvents** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **SubscribeToGUIEvents-RequestParams** – see 16.22.2) of the **subscribeToGUIEvents** request BIP message;

b) create a temporary abstract value (say, *subscribeToGUIEventsCallParams*) of type **SubscribeToGUIEventsCallParams** (see 16.22.3) where:

1) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of *incomingRequestParams* (presence and value);

2) if the optional component **bspProductUuid** of *incomingRequestParams* is present, then the component **bspUuid** of *subscribeToGUIEventsCallParams* shall be set from that component; otherwise, it shall be absent;

3) if the optional component **originalBSPHandle** of *incomingRequestParams* is present, then the component **bspHandle** of *subscribeToGUIEventsCallParams* shall be set from that component; otherwise, it shall be absent;

4) if the component **guiSelectEventSubscribed** of *incomingRequestParams* has the value **TRUE**, then the component **guiSelectEventHandlerAddress** shall be set to an implementation-defined memory address different from 0; otherwise, it shall be set to 0;

NOTE 1 – The actual memory address assigned to **guiSelectEventHandlerAddress** does not matter, because no calls will ever be made to that address (see 17.2.5).

5) if the component **guiStateEventSubscribed** of *incomingRequestParams* has the value **TRUE**, then the component **guiStateEventHandlerAddress** shall be set to an implementation-defined memory address different from 0; otherwise, it shall be set to 0;

NOTE 2 – The actual memory address assigned to **guiStateEventHandlerAddress** does not matter, because no calls will ever be made to that address (see 17.3.5).

6) if the component **guiProgressEventSubscribed** of *incomingRequestParams* has the value **TRUE**, then the component **guiProgressEventHandlerAddress** shall be set to an implementation-defined memory address different from 0; otherwise, it shall be set to 0; and

NOTE 3 – The actual memory address assigned to **guiProgressEventHandlerAddress** does not matter, because no calls will ever be made to that address (see 17.4.5).

7) the components **guiSelectEventHandlerContext**, **guiStateEventHandlerContext**, and **guiProgressEventHandlerContext** shall be set to 0;

c) if the components **bspProductUuid** and **originalBSPHandle** of *incomingRequestParams* are both present, then create and send a corresponding **subscribeToGUIEvents** response BIP message (see 13.3) with the return value set to **BioAPIERR_UUID_AND_HANDLE_BOTH_PRESENT**, skipping the remaining actions;

d) if the components **bspProductUuid** and **originalBSPHandle** of *incomingRequestParams* are both absent, then create and send a corresponding **subscribeToGUIEvents** response BIP message (see 13.3) with the return value set to **BioAPIERR_UUID_AND_HANDLE_BOTH_ABSENT**, skipping the remaining actions;

e) make an internal BioAPI function call (see 13.10) to the function **BioAPI_SubscribeToGUIEvents**, where the parameters of the function call shall be set by converting from *subscribeToGUIEventsCallParams* as specified in 16.22.6;

f) if the return value of the internal call is not 0, then create and send a corresponding **subscribeToGUIEvents** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

g) add an entry to the **GUIEventRemoteSubscriptions** table (see 18.11), where:

1) the component **subscriberEndpointIRI** shall be set to the master endpoint IRI;

2) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of *incomingRequestParams* (presence and value);

3) if the optional component **bspProductUuid** of *incomingRequestParams* is present, then the component **bspProductUuid** of the entry shall be set from that component; otherwise, it shall be set from the component **bspProductUuid** of the entry of the **AttachSessionRemoteReferences** table (see 18.9) where the component **originalBSPHandle** has the same value as the component **originalBSPHandle** of *incomingRequestParams*;

4) if the optional component **originalBSPHandle** of *incomingRequestParams* is present, then the component **bspHandle** of the entry shall be set from that component; otherwise, it shall be absent; and

5) the components **guiSelectEventSubscribed**, **guiStateEventSubscribed**, and **guiProgressEventSubscribed** shall be set from the components of *incomingRequestParams* with the same names;

h) create and send a corresponding **subscribeToGUIEvents** response BIP message (see 13.3) with the parameter value set to **NULL** and the return value set to 0.

16.22.6 Conversion between the parameters of the C function **BioAPI_SubscribeToGUIEvents** and the ASN.1 type **SubscribeToGUIEventsCallParams** (see 16.22.3) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 62.

Table 62 – Mapping between the parameters of the function BioAPI_SubscribeToGUIEvents and the ASN.1 type SubscribeToGUIEventsCallParams

Function parameter	Component of the ASN.1 type	References
GUIEventSubscriptionUuid	guiEventSubscriptionUuid	clause 19 in conjunction with 15.58
BSPUuid	bspUuid	clause 19 in conjunction with 15.58
BSPHandle	bspHandle	clause 19 in conjunction with 15.42
GUISelectEventHandler	guiSelectEventHandlerAddress	15.1.7
GUISelectEventHandler	guiSelectEventHandlerAddress	15.1.7
GUIStateEventHandler	guiStateEventHandlerAddress	15.1.7
GUIStateEventHandler	guiStateEventHandlerAddress	15.1.7
GUIProgressEventHandler	guiProgressEventHandlerAddress	15.1.7
GUIProgressEventHandlerCtx	guiProgressEventHandlerContext	15.1.7

16.23 Function BioAPI_UnsubscribeFromGUIEvents

16.23.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_UnsubscribeFromGUIEvents
        (const BioAPI_UUID *GUIEventSubscriptionUuid,
        const BioAPI_UUID *BSPUuid,
        const BioAPI_HANDLE *BSPHandle,
        BioAPI_GUI_SELECT_EVENT_HANDLER GUISelectEventHandler,
        void *GUISelectEventHandlerCtx,
        BioAPI_GUI_STATE_EVENT_HANDLER GUIStateEventHandler,
        void *GUIStateEventHandlerCtx,
        BioAPI_GUI_PROGRESS_EVENT_HANDLER GUIProgressEventHandler,
        void *GUIProgressEventHandlerCtx);
```

16.23.2 A pair of BIP message types are related to this function: the **unsubscribeFromGUIEvents** request BIP message type and the **unsubscribeFromGUIEvents** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
UnsubscribeFromGUIEvents-RequestParams ::= SEQUENCE {
        guiEventSubscriptionUuid        BioAPI-UUID OPTIONAL,
        bspProductUuid                  BioAPI-UUID OPTIONAL,
        originalBSPHandle               BioAPI-HANDLE OPTIONAL,
        guiSelectEventSubscribed        BOOLEAN,
        guiStateEventSubscribed         BOOLEAN,
        guiProgressEventSubscribed      BOOLEAN
    }
```

and:

```
UnsubscribeFromGUIEvents-ResponseParams ::= NULL
```

16.23.3 The following ASN.1 type is defined to aid the specification of the behaviour of a framework, but its abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
UnsubscribeFromGUIEventsCallParams ::= SEQUENCE {
        guiEventSubscriptionUuid        BioAPI-UUID OPTIONAL,
        bspUuid                         BioAPI-UUID OPTIONAL,
        bspHandle                       BioAPI-HANDLE OPTIONAL,
        guiSelectEventHandlerAddress    MemoryAddress,
        guiSelectEventHandlerContext    MemoryAddress,
        guiStateEventHandlerAddress     MemoryAddress,
        guiStateEventHandlerContext     MemoryAddress,
        guiProgressEventHandlerAddress  MemoryAddress,
        guiProgressEventHandlerContext  MemoryAddress
    }
```

16.23.4 When a framework receives a call to the function **BioAPI_UnsubscribeFromGUIEvents** from the local application, it shall first determine either:

a) the hosting endpoint and the BSP product UUID (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23 (if the parameter **BSPUuid** has a value different from **NULL**); or

b) the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the C variable pointed to by the parameter **BSPHandle** as specified in clause 24 (if the parameter **BSPHandle** has a value different from **NULL**);

and then one of the three following subclauses applies.

NOTE – Exactly one of these two parameters is required to have the value **NULL** (see ISO/IEC 19784-1, 8.3.8.1).

16.23.4.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that:

1) if the parameter **BSPUuid** of the incoming call had a value different from **NULL**, then the parameter **BSPUuid** shall be set by converting from *bspProductUuid* as specified in clause 19 in conjunction with 15.58; and

2) if the parameter **BSPHandle** of the incoming call had a value different from **NULL**, then the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in clause 19 in conjunction with 15.42;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create a temporary abstract value (say, *unsubscribeFromGUIEventsCallParams*) of type **UnsubscribeFromGUIEventsCallParams** (see 16.23.3) by converting from the parameters of the **BioAPI_UnsubscribeFromGUIEvents** function call as specified in 16.23.6;

d) search the **GUIEventLocalSubscriptions** table (see 18.10) for an entry where:

1) the optional component **guiEventSubscriptionUuid** has the same presence and value as the optional component **guiEventSubscriptionUuid** of *unsubscribeFromGUIEventsCallParams*;

2) the component **hostingEndpointIRI** contains the local endpoint IRI;

3) if the optional component **bspUuid** of *unsubscribeFromGUIEventsCallParams* is present, then the component **bspProductUuid** of the entry has the value *bspProductUuid*; otherwise, it has the same value as the component **bspProductUuid** of the entry of the **AttachSessionLocalReferences** table (see 18.8) where the component **originalBSPHandle** has the value *originalBSPHandle*;

4) if the optional component **bspUuid** of *unsubscribeFromGUIEventsCallParams* is absent or has a value different from *bspProductUuid*, then the component **useBSPAccessUuid** of the entry has the value **TRUE**; otherwise, it has the value **FALSE**;

5) if the optional component **bspHandle** of *unsubscribeFromGUIEventsCallParams* is present, then the optional component **originalBSPHandle** of the entry is present and has the value *originalBSPHandle*; otherwise, it is absent; and

6) the components **guiSelectEventHandlerAddress**, **guiSelectEventHandlerContext**, **guiStateEventHandlerAddress**, **guiStateEventHandlerContext**, **guiProgressEventHandlerAddress**, and **guiProgressEventHandlerContext** have the same values as the components of *unsubscribeFromGUIEventsCallParams* with the same names;

e) if there is no such entry, then return the value **BioAPIERR_NO_SUCH_SUBSCRIPTION_FOUND** to the local application, skipping the remaining actions;

f) delete the entry of the **GUIEventLocalSubscriptions** table (subclause 18.10.3 applies);

g) return the value 0 to the local application.

16.23.4.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *unsubscribeFromGUIEventsCallParams*) of type **UnsubscribeFromGUIEventsCallParams** (see 16.23.3) by converting from the parameters of the **BioAPI_UnsubscribeFromGUIEvents** function call as specified in 16.23.6;

b) create a temporary abstract value (say, *outgoingRequestParams*) of type **UnsubscribeFromGUIEvents-RequestParams** (see 16.23.2) where:

1) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of *unsubscribeFromGUIEventsCallParams* (presence and value);

2) if the optional component **bspUuid** of *unsubscribeFromGUIEventsCallParams* is present, then the component **bspProductUuid** of *outgoingRequestParams* shall be set to *bspProductUuid*; otherwise, it shall be absent;

3) if the optional component **bspHandle** of *unsubscribeFromGUIEventsCallParams* is present, then the optional component **originalBSPHandle** of *outgoingRequestParams* shall be set to *originalBSPHandle*; otherwise, it shall be absent;

4) if the component **guiSelectEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* has a value different from 0, then the component **guiSelectEventSubscribed** of *outgoingRequestParams* shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

5) if the component **guiStateEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* has a value different from 0, then the component **guiStateEventSubscribed** of *outgoingRequestParams* shall be set to **TRUE**; otherwise, it shall be set to **FALSE**; and

6) if the component **guiProgressEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* has a value different from 0, then the component **guiProgressEventSubscribed** of *outgoingRequestParams* shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

c) create and send an **unsubscribeFromGUIEvents** request BIP message (see 13.2) with the slave endpoint IRI set to the hosting endpoint IRI and the parameter value set to *outgoingRequestParams*;

d) receive a corresponding **unsubscribeFromGUIEvents** response BIP message (see 13.6);

e) if the return value of the response BIP message is not 0, then return that value to the local application, skipping the remaining actions;

f) search the **GUIEventLocalSubscriptions** table (see 18.10) for an entry where:

1) the optional component **guiEventSubscriptionUuid** has the same presence and value as the optional component **guiEventSubscriptionUuid** of *unsubscribeFromGUIEventsCallParams*;

2) the component **hostingEndpointIRI** contains the hosting endpoint IRI;

3) if the optional component **bspUuid** of *unsubscribeFromGUIEventsCallParams* is present, then the component **bspProductUuid** of the entry has the value *bspProductUuid*; otherwise, it has the same value as the component **bspProductUuid** of the entry of the **AttachSessionLocalReferences** table (see 18.8) where the component **originalBSPHandle** has the value *originalBSPHandle*;

4) if the optional component **bspUuid** of *unsubscribeFromGUIEventsCallParams* is absent or has a value different from *bspProductUuid*, then the component **useBSPAccessUuid** of the entry has the value **TRUE**; otherwise, it has the value **FALSE**;

5) if the optional component **bspHandle** of *unsubscribeFromGUIEventsCallParams* is present, then the optional component **originalBSPHandle** of the entry is present and has the value *originalBSPHandle*; otherwise, it is absent; and

6) the components **guiSelectEventHandlerAddress**, **guiSelectEventHandlerContext**, **guiStateEventHandlerAddress**, **guiStateEventHandlerContext**, **guiProgressEventHandlerAddress**, and **guiProgressEventHandlerContext** have the same values as the components of *unsubscribeFromGUIEventsCallParams* with the same names;

g) if there is no such entry, then return the value **BioAPIERR_NO_SUCH_SUBSCRIPTION_FOUND** to the local application, skipping the remaining actions;

h) delete the entry of the **GUIEventLocalSubscriptions** table (subclause 18.10.3 applies);

i) return the value 0 to the local application.

16.23.4.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.23.5 When a framework receives (see 13.9) an **unsubscribeFromGUIEvents** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **UnsubscribeFromGUIEvents-RequestParams** – see 16.23.2) of the **unsubscribeFromGUIEvents** request BIP message;

b) create a temporary abstract value (say, *unsubscribeFromGUIEventsCallParams*) of type **UnsubscribeFromGUIEventsCallParams** (see 16.23.3), where:

1) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of *incomingRequestParams* (presence and value);

2) if the optional component **bspProductUuid** of *incomingRequestParams* is present, then the optional component **bspUuid** shall be set from that component; otherwise, it shall be absent;

3) the optional component **bspHandle** shall be set from the optional component **originalBSPHandle** of *incomingRequestParams* (presence and value);

4) if the component **guiSelectEventSubscribed** of *incomingRequestParams* has the value **TRUE**, then the component **guiSelectEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

5) if the component **guiStateEventSubscribed** of *incomingRequestParams* has the value **TRUE**, then the component **guiStateEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

6) if the component **guiProgressEventSubscribed** of *incomingRequestParams* has the value **TRUE**, then the component **guiProgressEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0; and

7) the components **guiSelectEventHandlerContext**, **guiStateEventHandlerContext** and **guiProgressEventHandlerContext** shall be set to 0;

c) if the components **bspProductUuid** and **originalBSPHandle** of *incomingRequestParams* are both present, then create and send a corresponding **unsubscribeFromGUIEvents** response BIP message (see 13.3) with the return value set to **BioAPIERR_UUID_AND_HANDLE_BOTH_PRESENT**, skipping the remaining actions;

d) if the components **bspProductUuid** and **originalBSPHandle** of *incomingRequestParams* are both absent, then create and send a corresponding **unsubscribeFromGUIEvents** response BIP message (see 13.3) with the return value set to **BioAPIERR_UUID_AND_HANDLE_BOTH_ABSENT**, skipping the remaining actions;

e) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnsubscribeFromGUIEvents**, where the parameters of the function call shall be set by converting from *unsubscribeFromGUIEventsCallParams* as specified in 16.23.6;

f) if the return value of the internal call is not 0, then create and send a corresponding **unsubscribeFromGUIEvents** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

g) search the **GUIEventRemoteSubscriptions** table (see 18.11) for an entry where:

1) the component **subscriberEndpointIRI** contains the master endpoint IRI;

2) the optional component **guiEventSubscriptionUuid** has the same presence and value as the optional component **guiEventSubscriptionUuid** of *incomingRequestParams*;

3) if the optional component **bspProductUuid** of *incomingRequestParams* is present, then the component **bspProductUuid** of the entry has the same value as that component; otherwise, it has the same value as the component **bspProductUuid** of the entry of the **AttachSessionRemoteReferences** table (see 18.9) where the component **originalBSPHandle** has the same value as the component **originalBSPHandle** of *incomingRequestParams*;

4) if the optional component **originalBSPHandle** of *incomingRequestParams* is present, then the component **bspHandle** of the entry has the same value as that component; otherwise, it is absent; and

5) the components **guiSelectEventSubscribe**, **guiStateEventSubscribe**, and **guiProgressEventSubscribed** have the same values as the components of *incomingRequestParams* with the same names;

h) if there is no such entry, then create and send a corresponding **unsubscribeFromGUIEvents** response BIP message (see 13.3) with the return value set to **BioAPIERR_NO_SUCH_SUBSCRIPTION_FOUND**, skipping the remaining actions;

i) delete the entry of the **GUIEventRemoteSubscriptions** table (subclause 18.11.3 applies);

j) create and send a corresponding **unsubscribeFromGUIEvents** response BIP message (see 13.3) with the parameter value set to **NULL** and the return value set to 0.

16.23.6 Conversion between the parameters of the C function **BioAPI_UnsubscribeFromGUIEvents** and the ASN.1 type **UnsubscribeFromGUIEventsCallParams** (see 16.23.3) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 63.

Table 63 – Mapping between the parameters of the function BioAPI_UnsubscribeFromGUIEvents and the ASN.1 type UnsubscribeFromGUIEventsCallParams

Function parameter	Component of the ASN.1 type	References
GUIEventSubscriptionUuid	guiEventSubscriptionUuid	clause 19 in conjunction with 15.58
BSPUuid	bspUuid	clause 19 in conjunction with 15.58
BSPHandle	bspHandle	clause 19 in conjunction with 15.42
GUISelectEventHandler	guiSelectEventHandlerAddress	15.1.7
GUISelectEventHandler	guiSelectEventHandlerAddress	15.1.7
GUIStateEventHandler	guiStateEventHandlerAddress	15.1.7
GUIStateEventHandler	guiStateEventHandlerAddress	15.1.7
GUIProgressEventHandler	guiProgressEventHandlerAddress	15.1.7
GUIProgressEventHandlerCtx	guiProgressEventHandlerContext	15.1.7

16.24 Function BioAPI_QueryGUIEventSubscriptions

16.24.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_QueryGUIEventSubscriptions
        (const BioAPI_UUID *BSPUuid,
        BioAPI_GUI_EVENT_SUBSCRIPTION **GUIEventSubscriptionList,
        uint32_t *NumberOfElements);
```

16.24.2 A pair of BIP message types are related to this function: the **queryGUIEventSubscriptions** request BIP message type and the **queryGUIEventSubscriptions** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
QueryGUIEventSubscriptions-RequestParams ::= SEQUENCE {
        bspProductUuid          BioAPI-UUID
    }
```

and:

```
QueryGUIEventSubscriptions-ResponseParams ::= SEQUENCE {
        guiEventSubscriptions   SEQUENCE (SIZE(0..max-unsigned-int)) OF
                                subscription BioAPI-GUI-EVENT-SUBSCRIPTION
    }
```

16.24.3 When a framework receives a call to the function **BioAPI_QueryGUIEventSubscriptions** from the local application, it shall first determine the hosting endpoint and the product UUID of the BSP (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23, and then one of the three following subclauses applies.

16.24.3.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *outgoingResponseParams*) of type **QueryGUIEventSubscriptions-ResponseParams** (see 16.24.2), where the component **guiEventSubscriptions** shall be initially empty;

b) search the **GUIEventLocalSubscriptions** table (see 18.10) for all entries where:

   1) the optional component **guiEventSubscriptionUuid** is present;

   2) the component **hostingEndpointIRI** contains the local endpoint IRI; and

   3) the component **bspProductUuid** has the value *bspProductUuid*;

c) for each such entry (say, *localSubscription*), add an element to the component **guiEventSubscriptions** of *outgoingResponseParams*, where:

   1) the component **subscriberEndpointIRI** shall be set to the local endpoint IRI;

   2) the component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of *localSubscription*;

   3) if the component **guiSelectEventHandlerAddress** of *localSubscription* has a value different from 0, then the component **guiSelectEventSubscribed** shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

   4) the component **guiStateEventHandlerAddress** of *localSubscription* has a value different from 0, then the component **guiStateEventSubscribed** shall be set to **TRUE**; otherwise, it shall be set to **FALSE**; and

   5) if the component **guiProgressEventHandlerAddress** of *localSubscription* has a value different from 0, then the component **guiProgressEventSubscribed** shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

d) search the **GUIEventRemoteSubscriptions** table (see 18.11) for all entries where the optional component **guiEventSubscriptionUuid** is present and the component **bspProductUUid** has the value *bspProductUuid*;

e) for each such entry (say, *remoteSubscription*), add an element to the component **guiEventSubscriptions** of *outgoingResponseParams*, where all the components shall be set from the components of *remoteSubscription* with the same names;

f) set the output parameters of the **BioAPI_QueryGUIEventSubscriptions** function call by converting from *outgoingResponseParams* as specified in 16.24.6;

g) return the value 0 to the local application.

16.24.3.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **queryGUIEventSubscriptions** request/response BIP message pair as specified in clause 27, using 16.24.5 and 16.24.6 to convert between function parameters and ASN.1 components when required within that clause.

16.24.3.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.24.4 When a framework receives (see 13.9) a **queryGUIEventSubscriptions** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **QueryGUIEventSubscriptions-RequestParams** – see 16.24.2) of the **queryGUIEventSubscriptions** request BIP message;

b) create a temporary abstract value (say, *outgoingResponseParams*) of type **QueryGUIEventSubscriptions-ResponseParams** (see 16.24.2), where the component **guiEventSubscriptions** shall be initially empty;

c) search the **GUIEventLocalSubscriptions** table (see 18.10) for all entries where:

   1) the optional component **guiEventSubscriptionUuid** is present;

   2) the component **hostingEndpointIRI** contains the local endpoint IRI; and

   3) the component **bspProductUuid** has the same value as the component **bspProductUuid** of *incomingRequestParams*;

d) for each such entry (say, *localSubscription*), add an element to the component **guiEventSubscriptions** of *outgoingResponseParams*, where:

   1) the component **subscriberEndpointIRI** shall be set to the local endpoint IRI;

   2) the component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of *localSubscription*;

   3) if the component **guiSelectEventHandlerAddress** of *localSubscription* has a value different from 0, then the component **guiSelectEventSubscribed** shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

   4) if the component **guiStateEventHandlerAddress** of *localSubscription* has a value different from 0, then the component **guiStateEventSubscribed** shall be set to **TRUE**; otherwise, it shall be set to **FALSE**; and

   5) if the component **guiProgressEventHandlerAddress** of *localSubscription* has a value different from 0, then the component **guiProgressEventSubscribed** shall be set to **TRUE**; otherwise, it shall be set to **FALSE**;

e) search the **GUIEventRemoteSubscriptions** table (see 18.11) for all entries where the optional component **guiEventSubscriptionUuid** is present and the component **bspProductUuid** has the same value as the component **bspProductUuid** of *incomingRequestParams*;

f) for each such entry (say, *remoteSubscription*), add an element to the component **guiEventSubscriptions** of *outgoingResponseParams*, where all the components shall be set from the components of *remoteSubscription* with the same names;

g) create and send a corresponding **queryGUIEventSubscriptions** response BIP message (see 13.3) with the parameter value set to *outgoingResponseParams* and the return value set to 0.

16.24.5 Conversion between the parameters of the C function **BioAPI_QueryGUIEventSubscriptions** and the ASN.1 type **QueryGUIEventSubscriptions-RequestParams** (see 16.24.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 64.

Table 64 – Mapping between the parameters of the function BioAPI_QueryGUIEventSubscriptions and the ASN.1 type QueryGUIEventSubscriptions-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPUuid	bspProductUuid	clause 25
GUIEventSubscriptionList	*none*	clause 22
NumberOfElements	*none*	clause 22

16.24.6 Conversion between the parameters of the C function **BioAPI_QueryGUIEventSubscriptions** and the ASN.1 type **QueryGUIEventSubscriptions-ResponseParams** (see 16.24.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 65.

Table 65 – Mapping between the parameters of the function BioAPI_QueryGUIEventSubscriptions and the ASN.1 type QueryGUIEventSubscriptions-ResponseParams

Function parameter	Component of the ASN.1 type	References
GUIEventSubscriptionList, NumberOfElements	guiEventSubscriptions	clause 20 in conjunction with 16.24.7 and 16.24.8

16.24.7 Conversion from the pair of C variables pointed to by the parameters **GUIEventSubscriptionList**/**NumberOfElements** to the ASN.1 component **guiEventSubscriptions** shall be done as follows: Calling *N* the value of the C variable pointed to by the parameter **NumberOfElements**, each of the first *N* elements (of type **BioAPI_GUI_EVENT_SUBSCRIPTION** – see 15.36) in the array pointed to by the C variable pointed to by the parameter **GUIEventSubscriptionList** shall be converted, in order, to an element of the component **guiEventSubscriptions** as specified in 15.36. The component **guiEventSubscriptions** shall have exactly *N* elements.

16.24.8 Conversion from the ASN.1 component **guiEventSubscriptions** to the pair of C variables pointed to by the parameters **GUIEventSubscriptionList**/**NumberOfElements** shall be done as follows: Calling *N* the number of elements of the component **guiEventSubscriptions**, a newly allocated array of *N* elements of type **BioAPI_GUI_EVENT_SUBSCRIPTION** (see 15.36) shall be filled by converting each element of the component **guiEventSubscriptions**, in order, to an element of the array as specified in 15.36. The C variable pointed to by the parameter **GUIEventSubscriptionList** shall be set to the address of the array, and the C variable pointed to by the parameter **NumberOfElements** shall be set to *N*.

16.25 Function BioAPI_NotifyGUISelectEvent

16.25.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_NotifyGUISelectEvent
        (const uint8_t *SubscriberEndpointIRI,
        const BioAPI_UUID *GUIEventSubscriptionUuid,
        const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_ID UnitID,
        BioAPI_GUI_ENROLL_TYPE EnrollType,
        BioAPI_GUI_OPERATION Operation,
        BioAPI_GUI_MOMENT Moment,
        BioAPI_RETURN ResultCode,
        uint32_t MaxNumEnrollSamples,
        BioAPI_BIR_SUBTYPE_MASK SelectableInstances,
        BioAPI_BIR_SUBTYPE_MASK *SelectedInstances,
        BioAPI_BIR_SUBTYPE_MASK CapturedInstances,
        const uint8_t *Text,
        BioAPI_GUI_RESPONSE *Response);
```

16.25.2 A pair of BIP message types are related to this function: the **notifyGUISelectEvent** request BIP message type and the **notifyGUISelectEvent** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
NotifyGUISelectEvent-RequestParams ::= SEQUENCE {
        subscriberEndpointIRI           EndpointIRI,
        guiEventSubscriptionUuid        BioAPI-UUID,
        bspProductUuid                  BioAPI-UUID,
        unitID                          BioAPI-UNIT-ID,
        enrollType                      BioAPI-GUI-ENROLL-TYPE,
        operation                       BioAPI-GUI-OPERATION,
        moment                          BioAPI-GUI-MOMENT,
        resultCode                      BioAPI-RETURN,
        maxNumEnrollSamples             UnsignedInt,
        selectableInstances             BioAPI-BIR-SUBTYPE-MASK,
        capturedInstances               BioAPI-BIR-SUBTYPE-MASK,
        text                            UTF8String OPTIONAL
    }
```

and:

```
NotifyGUISelectEvent-ResponseParams ::= SEQUENCE {
        selectedInstances           BioAPI-BIR-SUBTYPE-MASK,
        response                    BioAPI-GUI-RESPONSE
    }
```

16.25.3 When a framework receives a call to the function **BioAPI_NotifyGUISelectEvent** from the local application, it shall first determine the hosting endpoint and the product UUID of the BSP (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23, and then one of the three following subclauses applies.

16.25.3.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *incomingRequestParams*) of type **NotifyGUISelectEvent-RequestParams** (see 16.25.2) by converting from the parameters of the **BioAPI_NotifyGUISelectEvent** function call as specified in 16.25.5;

b) create a temporary abstract value (say, *eventInfo*) of type **GUISelectEventInfo** (see 17.2.4) where:

   1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

   2) the optional component **originalBSPHandle** shall be absent; and

   3) the remaining components shall be set from the components of *incomingRequestParams* with the same names;

c) notify a GUI select event based on *eventInfo* to a subscriber (either a GUI select event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 30;

d) if *incomingReturnValue* is not 0, then return that value to the local application, skipping the remaining actions;

e) create a temporary abstract value (say, *outgoingResponseParams*) of type **NotifyGUISelectEvent-ResponseParams** (see 16.25.2), where all the components shall be set from the components of *incomingAcknowledgementParams* with the same names;

f) set the output parameters of the **BioAPI_NotifyGUISelectEvent** function call by converting from *outgoingResponseParams* as specified in 16.25.6; and

g) return the value 0 to the local application.

16.25.3.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **notifyGUISelectEvent** request/response BIP message pair as specified in clause 27, using 16.25.5 and 16.25.6 to convert between function parameters and ASN.1 components when required within that clause.

16.25.3.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.25.4 When a framework receives (see 13.9) a **notifyGUISelectEvent** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **NotifyGUISelectEvent-RequestParams** – see 16.25.2) of the **notifyGUISelectEvent** request BIP message;

b) create a temporary abstract value (say, *eventInfo*) of type **GUISelectEventInfo** (see 17.2.4) where:

   1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

   2) the optional component **originalBSPHandle** shall be absent; and

   3) the remaining components shall be set from the components of *incomingRequestParams* with the same names;

c) notify a GUI select event based on *eventInfo* to a subscriber (either a GUI select event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 30;

d) if *incomingReturnValue* is not 0, then create and send a corresponding **notifyGUISelectEvent** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

e) create a temporary abstract value (say, *outgoingResponseParams*) of type **NotifyGUISelectEvent-ResponseParams** (see 16.25.2), where all the components shall be set from the components of *incomingAcknowledgementParams* with the same names;

f) create and send a corresponding **notifyGUISelectEvent** response BIP message (see 13.3) with the parameter value set to *outgoingResponseParams* and the return value set to 0.

16.25.5 Conversion between the parameters of the C function **BioAPI_NotifyGUISelectEvent** and the ASN.1 type **NotifyGUISelectEvent-RequestParams** (see 16.25.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 66.

Table 66 – Mapping between the parameters of the function BioAPI_NotifyGUISelectEvent and the ASN.1 type NotifyGUISelectEvent-RequestParams

Function parameter	Component of the ASN.1 type	References
SubscriberEndpointIRI	subscriberEndpointIRI	15.3
GUIEventSubscriptionUuid	guiEventSubscriptionUuid	clause 19 in conjunction with 15.58
BSPUuid	bspProductUuid	clause 25
UnitID	unitID	15.55
EnrollType	enrollType	15.38
Operation	operation	15.39
Moment	moment	15.37
ResultCode	resultCode	15.52
MaxNumEnrollSamples	maxNumEnrollSamples	15.1.6
SelectableInstances	selectableInstances	15.17
CapturedInstances	capturedInstances	15.17
Text	text	15.2
SelectedInstances	*none*	clause 22
Response	*none*	clause 22

16.25.6 Conversion between the parameters of the C function **BioAPI_NotifyGUISelectEvent** and the ASN.1 type **NotifyGUISelectEvent-ResponseParams** (see 16.25.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 67.

Table 67 – Mapping between the parameters of the function BioAPI_NotifyGUISelectEvent and the ASN.1 type NotifyGUISelectEvent-ResponseParams

Function parameter	Component of the ASN.1 type	References
SelectedInstances	selectedInstances	clause 20 in conjunction with 15.17
Response	response	clause 20 in conjunction with 15.40

16.26 Function BioAPI_NotifyGUIStateEvent

16.26.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_NotifyGUIStateEvent
        (const uint8_t *SubscriberEndpointIRI,
        const BioAPI_UUID *GUIEventSubscriptionUuid,
        const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_ID UnitID,
        BioAPI_GUI_OPERATION Operation,
        BioAPI_GUI_SUBOPERATION Suboperation,
        BioAPI_BIR_PURPOSE Purpose,
        BioAPI_GUI_MOMENT Moment,
        BioAPI_RETURN ResultCode,
        int32_t EnrollSampleIndex,
        const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
        const uint8_t *Text,
        BioAPI_GUI_RESPONSE *Response,
        int32_t *EnrollSampleIndexToRecapture);
```

16.26.2 A pair of BIP message types are related to this function: the **notifyGUIStateEvent** request BIP message type and the **notifyGUIStateEvent** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
NotifyGUIStateEvent-RequestParams ::= SEQUENCE {
        subscriberEndpointIRI           EndpointIRI,
        guiEventSubscriptionUuid        BioAPI-UUID,
        bspProductUuid                  BioAPI-UUID,
        unitID                          BioAPI-UNIT-ID,
        operation                       BioAPI-GUI-OPERATION,
        suboperation                    BioAPI-GUI-SUBOPERATION,
        purpose                         BioAPI-BIR-PURPOSE,
        moment                          BioAPI-GUI-MOMENT,
        resultCode                      BioAPI-RETURN,
        enrollSampleIndex               SignedInt,
        bitmaps                         BioAPI-GUI-BITMAP-ARRAY OPTIONAL,
        text                            UTF8String OPTIONAL
    }
```

and:

```
NotifyGUIStateEvent-ResponseParams ::= SEQUENCE {
        response                        BioAPI-GUI-RESPONSE,
        enrollSampleIndexToRecapture    SignedInt
    }
```

16.26.3 When a framework receives a call to the function **BioAPI_NotifyGUIStateEvent** from the local application, it shall first determine the hosting endpoint and the product UUID of the BSP (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23, and then one of the three following subclauses applies.

16.26.3.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *incomingRequestParams*) of type **NotifyGUIStateEvent-RequestParams** (see 16.26.2) by converting from the parameters of the **BioAPI_NotifyGUIStateEvent** function call as specified in 16.26.5;

b) create a temporary abstract value (say, *eventInfo*) of type **GUIStateEventInfo** (see 17.3.4) where:

   1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

   2) the optional component **originalBSPHandle** shall be absent; and

   3) the remaining components shall be set from the components of *incomingRequestParams* with the same names;

c) notify a GUI state event based on *eventInfo* to a subscriber (either a GUI state event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 31;

d) if *incomingReturnValue* is not 0, then return that value to the local application, skipping the remaining actions;

e) create a temporary abstract value (say, *outgoingResponseParams*) of type **NotifyGUIStateEvent-ResponseParams** (see 16.26.2), where all the components shall be set from the components of *incomingAcknowledgementParams* with the same names;

f) set the output parameters of the **BioAPI_NotifyGUIStateEvent** function call by converting from *outgoingResponseParams* as specified in 16.26.6; and

g) return the value 0 to the local application.

16.26.3.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **notifyGUIStateEvent** request/response BIP message pair as specified in clause 27, using 16.26.5 and 16.26.6 to convert between function parameters and ASN.1 components when required within that clause.

16.26.3.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.26.4 When a framework receives (see 13.9) a **notifyGUIStateEvent** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **NotifyGUIStateEvent-RequestParams** – see 16.26.2) of the **notifyGUIStateEvent** request BIP message;

b) create a temporary abstract value (say, *eventInfo*) of type **GUIStateEventInfo** (see 17.3.4) where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

2) the optional component **originalBSPHandle** shall be absent; and

3) the remaining components shall be set from the components of *incomingRequestParams* with the same names;

c) notify a GUI state event based on *eventInfo* to a subscriber (either a GUI state event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 31;

d) if *incomingReturnValue* is not 0, then create and send a corresponding **notifyGUIStateEvent** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

e) create a temporary abstract value (say, *outgoingResponseParams*) of type **NotifyGUIStateEvent-ResponseParams** (see 16.26.2), where all the components shall be set from the components of *incomingAcknowledgementParams* with the same names;

f) create and send a corresponding **notifyGUIStateEvent** response BIP message (see 13.3) with the parameter value set to *outgoingResponseParams* and the return value set to 0.

16.26.5 Conversion between the parameters of the C function **BioAPI_NotifyGUIStateEvent** and the ASN.1 type **NotifyGUIStateEvent-RequestParams** (see 16.26.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 68.

Table 68 – Mapping between the parameters of the function BioAPI_NotifyGUIStateEvent and the ASN.1 type NotifyGUIStateEvent-RequestParams

Function parameter	Component of the ASN.1 type	References
SubscriberEndpointIRI	subscriberEndpointIRI	15.3
GUIEventSubscriptionUuid	guiEventSubscriptionUuid	clause 19 in conjunction with 15.58
BSPUuid	bspProductUuid	clause 25
UnitID	unitID	15.55
Operation	operation	15.39
Suboperation	suboperation	15.41
Purpose	purpose	15.14
Moment	moment	15.37
ResultCode	resultCode	15.52
EnrollSampleIndex	enrollSampleIndex	15.1.6
Bitmaps	bitmaps	clause 19 in conjunction with 15.35
Text	text	15.2
Response	*none*	clause 22
EnrollSampleIndexToRecapture	*none*	clause 22

16.26.6 Conversion between the parameters of the C function **BioAPI_NotifyGUIStateEvent** and the ASN.1 type **NotifyGUIStateEvent-ResponseParams** (see 16.26.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 69.

Table 69 – Mapping between the parameters of the function BioAPI_NotifyGUIStateEvent and the ASN.1 type NotifyGUIStateEvent-ResponseParams

Function parameter	Component of the ASN.1 type	References
EnrollSampleIndexToRecapture	enrollSampleIndexToRecapture	clause 20 in conjunction with 15.1.6
Response	response	clause 20 in conjunction with 15.40

16.27 Function BioAPI_NotifyGUIProgressEvent

16.27.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_NotifyGUIProgressEvent
        (const uint8_t *SubscriberEndpointIRI,
        const BioAPI_UUID *GUIEventSubscriptionUuid,
        const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_ID UnitID,
        BioAPI_GUI_OPERATION Operation,
        BioAPI_GUI_SUBOPERATION Suboperation,
        BioAPI_BIR_PURPOSE Purpose,
        BioAPI_GUI_MOMENT Moment,
        uint8_t SuboperationProgress,
        const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
        const uint8_t *Text,
        BioAPI_GUI_RESPONSE *Response);
```

16.27.2 A pair of BIP message types are related to this function: the **notifyGUIProgressEvent** request BIP message type and the **notifyGUIProgressEvent** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
NotifyGUIProgressEvent-RequestParams ::= SEQUENCE {
        subscriberEndpointIRI           EndpointIRI,
        guiEventSubscriptionUuid        BioAPI-UUID,
        bspProductUuid                  BioAPI-UUID,
        unitID                          BioAPI-UNIT-ID,
        operation                       BioAPI-GUI-OPERATION,
        suboperation                    BioAPI-GUI-SUBOPERATION,
        purpose                         BioAPI-BIR-PURPOSE,
        moment                          BioAPI-GUI-MOMENT,
        suboperationProgress            UnsignedByte,
        bitmaps                         BioAPI-GUI-BITMAP-ARRAY OPTIONAL,
        text                            UTF8String OPTIONAL
    }
```

and:

```
NotifyGUIProgressEvent-ResponseParams ::= SEQUENCE {
        response                        BioAPI-GUI-RESPONSE
    }
```

16.27.3 When a framework receives a call to the function **BioAPI_NotifyGUIProgressEvent** from the local application, it shall first determine the hosting endpoint and the product UUID of the BSP (say, *bspProductUuid*) from the parameter **BSPUuid** as specified in clause 23, and then one of the three following subclauses applies.

16.27.3.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *incomingRequestParams*) of type **NotifyGUIProgressEvent-RequestParams** (see 16.27.2) by converting from the parameters of the **BioAPI_NotifyGUIProgressEvent** function call as specified in 16.27.5;

b) create a temporary abstract value (say, *eventInfo*) of type **GUIProgressEventInfo** (see 17.4.4) where:

   1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

   2) the optional component **originalBSPHandle** shall be absent; and

   3) the remaining components shall be set from the components of *incomingRequestParams* with the same names;

c) notify a GUI progress event based on *eventInfo* to a subscriber (either a GUI progress event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 32;

d) if *incomingReturnValue* is not 0, then return that value to the local application, skipping the remaining actions;

e) create a temporary abstract value (say, *outgoingResponseParams*) of type **NotifyGUIProgressEvent-ResponseParams** (see 16.27.2), where all the components shall be set from the components of *incomingAcknowledgementParams* with the same names;

f) set the output parameters of the **BioAPI_NotifyGUIProgressEvent** function call by converting from *outgoingResponseParams* as specified in 16.27.6; and

g) return the value 0 to the local application.

16.27.3.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **notifyGUIProgressEvent** request/response BIP message pair as specified in clause 27, using 16.27.5 and 16.27.6 to convert between function parameters and ASN.1 components when required within that clause.

16.27.3.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.27.4 When a framework receives (see 13.9) a **notifyGUIProgressEvent** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **NotifyGUIProgressEvent-RequestParams** – see 16.27.2) of the **notifyGUIProgressEvent** request BIP message;

b) create a temporary abstract value (say, *eventInfo*) of type **GUIProgressEventInfo** (see 17.4.4) where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

2) the optional component **originalBSPHandle** shall be absent; and

3) the remaining components shall be set from the components of *incomingRequestParams* with the same names;

c) notify a GUI progress event based on *eventInfo* to a subscriber (either a GUI progress event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 32;

d) if *incomingReturnValue* is not 0, then create and send a corresponding **notifyGUIProgressEvent** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

e) create a temporary abstract value (say, *outgoingResponseParams*) of type **NotifyGUIProgressEvent-ResponseParams** (see 16.27.2), where all the components shall be set from the components of *incomingAcknowledgementParams* with the same names;

f) create and send a corresponding **notifyGUIProgressEvent** response BIP message (see 13.3) with the parameter value set to *outgoingResponseParams* and the return value set to 0.

16.27.5 Conversion between the parameters of the C function **BioAPI_NotifyGUIProgressEvent** and the ASN.1 type **NotifyGUIProgressEvent-RequestParams** (see 16.27.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 70.

Table 70 – Mapping between the parameters of the function BioAPI_NotifyGUIProgressEvent and the ASN.1 type NotifyGUIProgressEvent-RequestParams

Function parameter	Component of the ASN.1 type	References
SubscriberEndpointIRI	subscriberEndpointIRI	15.3
GUIEventSubscriptionUuid	guiEventSubscriptionUuid	clause 19 in conjunction with 15.58
BSPUuid	bspProductUuid	clause 25
UnitID	unitID	15.55
Operation	operation	15.39
Suboperation	suboperation	15.41
Purpose	purpose	15.14
Moment	moment	15.37
SuboperationInProgress	suboperationProgress	15.1.3
Bitmaps	Bitmaps	clause 19 in conjunction with 15.35
Text	Text	15.2
Response	*None*	clause 22

16.27.6 Conversion between the parameters of the C function **BioAPI_NotifyGUIProgressEvent** and the ASN.1 type **NotifyGUIProgressEvent-ResponseParams** (see 16.27.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 71.

Table 71 – Mapping between the parameters of the function BioAPI_NotifyGUIProgressEvent and the ASN.1 type NotifyGUIProgressEvent-ResponseParams

Function parameter	Component of the ASN.1 type	References
Response	response	clause 20 in conjunction with 15.40

16.28 Function BioAPI_RedirectGUIEvents

16.28.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_RedirectGUIEvents
        (const uint8_t *SubscriberEndpointIRI,
        const BioAPI_UUID *GUIEventSubscriptionUuid,
        BioAPI_HANDLE BSPHandle,
        BioAPI_BOOL GUISelectEventRedirected,
        BioAPI_BOOL GUIStateEventRedirected,
        BioAPI_BOOL GUIProgressEventRedirected);
```

16.28.2 A pair of BIP message types are related to this function: the **redirectGUIEvents** request BIP message type and the **redirectGUIEvents** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
RedirectGUIEvents-RequestParams ::= SEQUENCE {
        subscriberEndpointIRI           EndpointIRI,
        guiEventSubscriptionUuid        BioAPI-UUID,
        originalBSPHandle               BioAPI-HANDLE,
        guiSelectEventRedirected        BOOLEAN,
        guiStateEventRedirected         BOOLEAN,
        guiProgressEventRedirected      BOOLEAN
}
```

and:

```
RedirectGUIEvents-ResponseParams ::= NULL
```

16.28.3 When a framework receives a call to the function **BioAPI_RedirectGUIEvents** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24, and then one of the three following subclauses applies.

16.28.3.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create a temporary abstract value (say, *incomingRequestParams*) of type **RedirectGUIEvents-RequestParams** (see 16.28.2) by converting from the parameters of the **BioAPI_RedirectGUIEvents** function call as specified in 16.28.5;

d) add an entry to the **GUIEventRedirectors** table (see 18.12), where:

   1) the components **referrerEndpointIRI** and **bspProductUuid** shall be set from the components with the same names of the entry of the **AttachSessionRemoteReferences** table (see 18.9) where the component **originalBSPHandle** has the same value as the component **originalBSPHandle** of *incomingRequestParams*; and

   2) the remaining components shall be set from the components of *incomingRequestParams* with the same names;

e) return the value 0 to the local application.

16.28.3.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **redirectGUIEvents** request/response BIP message pair as specified in clause 27, using 16.28.5 to convert between function parameters and ASN.1 components when required within that clause.

16.28.3.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.28.4 When a framework receives (see 13.9) a **redirectGUIEvents** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **RedirectGUIEvents-RequestParams** – see 16.28.2) of the **redirectGUIEvents** request BIP message;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_RedirectGUIEvents**, where the parameters of the function call shall be set by converting from *incomingRequestParams* as specified in 16.28.5;

c) if the return value of the internal call is not 0, then create and send a corresponding **redirectGUIEvents** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

d) add an entry to the **GUIEventRedirectors** table (see 18.12), where:

1) the components **referrerEndpointIRI** and **bspProductUuid** shall be set from the components with the same names of the entry of the **AttachSessionRemoteReferences** table (see 18.9) where the component **originalBSPHandle** has the same value as the component **originalBSPHandle** of *incomingRequestParams*; and

2) the remaining components shall be set from the components of *incomingRequestParams* with the same names;

e) create and send a corresponding **redirectGUIEvents** response BIP message (see 13.3) with the parameter value set to **NULL** and the return value set to 0.

16.28.5 Conversion between the parameters of the C function **BioAPI_RedirectGUIEvents** and the ASN.1 type **RedirectGUIEvents-RequestParams** (see 16.28.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 72.

Table 72 – Mapping between the parameters of the function BioAPI_RedirectGUIEvents and the ASN.1 type RedirectGUIEvents-RequestParams

Function parameter	Component of the ASN.1 type	References
SubscriberEndpointIRI	subscriberEndpointIRI	15.3
GUIEventSubscriptionUuid	guiEventSubscriptionUuid	clause 19 in conjunction with 15.58
BSPHandle	originalBSPHandle	clause 26
GUISelectEventRedirected	guiSelectEventRedirected	15.18
GUIStateEventRedirected	guiStateEventRedirected	15.18
GUIProgressEventRedirected	guiProgressEventRedirected	15.18

16.29 Function BioAPI_UnredirectGUIEvents

16.29.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_UnredirectGUIEvents
    (const uint8_t *SubscriberEndpointIRI,
     const BioAPI_UUID *GUIEventSubscriptionUuid,
     BioAPI_HANDLE BSPHandle,
     BioAPI_BOOL GUISelectEventRedirected,
     BioAPI_BOOL GUIStateEventRedirected,
     BioAPI_BOOL GUIProgressEventRedirected);
```

16.29.2 A pair of BIP message types are related to this function: the **unredirectGUIEvents** request BIP message type and the **unredirectGUIEvents** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
UnredirectGUIEvents-RequestParams ::= SEQUENCE {
        subscriberEndpointIRI           EndpointIRI,
        guiEventSubscriptionUuid        BioAPI-UUID,
        originalBSPHandle               BioAPI-HANDLE,
        guiSelectEventRedirected        BOOLEAN,
        guiStateEventRedirected         BOOLEAN,
        guiProgressEventRedirected      BOOLEAN
    }
```

and:

UnredirectGUIEvents-ResponseParams ::= NULL

16.29.3 When a framework receives a call to the function **BioAPI_UnredirectGUIEvents** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24, and then one of the three following subclauses applies.

16.29.3.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create a temporary abstract value (say, *incomingRequestParams*) of type **UnredirectGUIEvents-RequestParams** (see 16.29.2) by converting from the parameters of the **BioAPI_UnredirectGUIEvents** function call as specified in 16.29.5;

d) search the **GUIEventRedirectors** table (see 18.12) for an entry where the components **subscriberEndpointIRI**, **guiEventSubscriptionUuid**, **originalBSPHandle**, **guiSelectEventRedirected**, **guiStateEventRedirected**, and **guiProgressEventRedirected** have the same values as the components of *incomingRequestParams* with the same names;

e) if there is no such entry, then return the value **BioAPIERR_NO_SUCH_REDIRECTOR_FOUND** to the local application, skipping the remaining actions;

f) delete the entry of the **GUIEventRedirectors** table (subclause 18.12.3 applies);

NOTE – If multiple entries are found, any one of those entries (but exactly one) will be deleted.

g) return the value 0 to the local application.

16.29.3.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint an **unredirectGUIEvents** request/response BIP message pair as specified in clause 27, using 16.29.5 to convert between function parameters and ASN.1 components when required within that clause.

16.29.3.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.29.4 When a framework receives (see 13.9) an **unredirectGUIEvents** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **UnredirectGUIEvents-RequestParams** – see 16.29.2) of the **unredirectGUIEvents** request BIP message;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnredirectGUIEvents**, where the parameters of the function call shall be set by converting from *incomingRequestParams* as specified in 16.29.5;

c) if the return value of the internal call is not 0, then create and send a corresponding **unredirectGUIEvents** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

d) search the **GUIEventRedirectors** table (see 18.12) for an entry where the components **subscriberEndpointIRI**, **guiEventSubscriptionUuid**, **originalBSPHandle**, **guiSelectEventRedirected**, **guiStateEventRedirected**, and **guiProgressEventRedirected** have the same values as the components of *incomingRequestParams* with the same names;

e) if there is no such entry, then create and send a corresponding **unredirectGUIEvents** response BIP message (see 13.3) with the return value set to **BioAPIERR_NO_SUCH_REDIRECTOR_FOUND**, skipping the remaining actions;

f) delete the entry of the **GUIEventRedirectors** table (subclause 18.12.3 applies);

NOTE – If multiple entries are found, any one of those entries (but exactly one) will be deleted.

g) create and send a corresponding **unredirectGUIEvents** response BIP message (see 13.3) with the parameter value set to **NULL** and the return value set to 0.

16.29.5 Conversion between the parameters of the C function **BioAPI_UnredirectGUIEvents** and the ASN.1 type **UnredirectGUIEvents-RequestParams** (see 16.29.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 73.

Table 73 – Mapping between the parameters of the function BioAPI_UnredirectGUIEvents and the ASN.1 type UnredirectGUIEvents-RequestParams

Function parameter	Component of the ASN.1 type	References
SubscriberEndpointIRI	subscriberEndpointIRI	15.3
GUIEventSubscriptionUuid	guiEventSubscriptionUuid	clause 19 in conjunction with 15.58
BSPHandle	originalBSPHandle	clause 26
GUISelectEventRedirected	guiSelectEventRedirected	15.18
GUIStateEventRedirected	guiStateEventRedirected	15.18
GUIProgressEventRedirected	guiProgressEventRedirected	15.18

16.30 Function BioAPI_Capture

16.30.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Capture
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_BIR_PURPOSE Purpose,
        BioAPI_BIR_SUBTYPE Subtype,
        const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
        BioAPI_BIR_HANDLE *CapturedBIR,
        int32_t Timeout,
        BioAPI_BIR_HANDLE *AuditData);
```

16.30.2 A pair of BIP message types are related to this function: the **capture** request BIP message type and the **capture** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
Capture-RequestParams ::= SEQUENCE {
        originalBSPHandle           BioAPI-HANDLE,
        purpose                     BioAPI-BIR-PURPOSE,
        subtype                     BioAPI-BIR-SUBTYPE,
        outputFormat                BioAPI-BIR-BIOMETRIC-DATA-FORMAT OPTIONAL,
        timeout                     SignedInt,
        no-auditData                BOOLEAN
    }
```

and:

```
Capture-ResponseParams ::= SEQUENCE {
        capturedBIR                 BioAPI-BIR-HANDLE,
        auditData                   BioAPI-BIR-HANDLE OPTIONAL
    }
```

16.30.3 When a framework receives a call to the function **BioAPI_Capture** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **capture** request/response BIP message pair as specified in clause 27, using 16.30.5 and 16.30.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.30.4 When a framework receives (see 13.9) a **capture** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_Capture** to create and send a corresponding **capture** response BIP message as specified in clause 28, using 16.30.5 and 16.30.6 to convert between function parameters and ASN.1 components when required within that clause.

16.30.5 Conversion between the parameters of the C function **BioAPI_Capture** and the ASN.1 type **Capture-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 74.

Table 74 – Mapping between the parameters of the function BioAPI_Capture and the ASN.1 type Capture-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
Purpose	purpose	15.14
Subtype	subtype	15.16
OutputFormat	outputFormat	clause 19 in conjunction with 15.8
CapturedBIR	*none*	clause 22
Timeout	timeout	15.1.6
AuditData	no-auditData	clause 21

16.30.6 Conversion between the parameters of the C function **BioAPI_Capture** and the ASN.1 type **Capture-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 75.

Table 75 – Mapping between the parameters of the function BioAPI_Capture and the ASN.1 type Capture-ResponseParams

Function parameter	Component of the ASN.1 type	References
CapturedBIR	capturedBIR	clause 20 in conjunction with 15.12
AuditData	auditData	clause 20 in conjunction with 15.12

16.31 Function BioAPI_CreateTemplate

16.31.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_CreateTemplate
        (BioAPI_HANDLE  BSPHandle,
        const BioAPI_INPUT_BIR *CapturedBIR,
        const BioAPI_INPUT_BIR *ReferenceTemplate,
        const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
        BioAPI_BIR_HANDLE *NewTemplate,
        const BioAPI_DATA *Payload,
        BioAPI_UUID *TemplateUuid);
```

16.31.2 A pair of BIP message types are related to this function: the **createTemplate** request BIP message type and the **createTemplate** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
CreateTemplate-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        capturedBIR             BioAPI-INPUT-BIR,
        referenceTemplate       BioAPI-INPUT-BIR OPTIONAL,
        outputFormat            BioAPI-BIR-BIOMETRIC-DATA-FORMAT OPTIONAL,
        payload                 BioAPI-DATA OPTIONAL,
        no-templateUuid         BOOLEAN
    }
```

and:

```
CreateTemplate-ResponseParams ::= SEQUENCE {
        newTemplate             BioAPI-BIR-HANDLE,
        templateUuid            BioAPI-UUID OPTIONAL
    }
```

16.31.3 When a framework receives a call to the function **BioAPI_CreateTemplate** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework

shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **createTemplate** request/response BIP message pair as specified in clause 27, using 16.31.5 and 16.31.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.31.4 When a framework receives (see 13.9) a **createTemplate** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_CreateTemplate** to create and send a corresponding **createTemplate** response BIP message as specified in clause 28, using 16.31.5 and 16.31.6 to convert between function parameters and ASN.1 components when required within that clause.

16.31.5 Conversion between the parameters of the C function **BioAPI_CreateTemplate** and the ASN.1 type **CreateTemplate-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 76.

Table 76 – Mapping between the parameters of the function BioAPI_CreateTemplate and the ASN.1 type CreateTemplate-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
CapturedBIR	capturedBIR	clause 19 in conjunction with 15.46
ReferenceTemplate	referenceTemplate	clause 19 in conjunction with 15.46
OutputFormat	outputFormat	clause 19 in conjunction with 15.8
NewTemplate	*none*	clause 22
Payload	payload	clause 19 in conjunction with 15.22
TemplateUuid	no-templateUuid	clause 21

16.31.6 Conversion between the parameters of the C function **BioAPI_CreateTemplate** and the ASN.1 type **CreateTemplate-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 77.

Table 77 – Mapping between the parameters of the function BioAPI_CreateTemplate and the ASN.1 type CreateTemplate-ResponseParams

Function parameter	Component of the ASN.1 type	References
NewTemplate	newTemplate	clause 20 in conjunction with 15.12
TemplateUuid	templateUuid	clause 20 in conjunction with 15.58

16.32 Function BioAPI_Process

16.32.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Process
        (BioAPI_HANDLE  BSPHandle,
        const BioAPI_INPUT_BIR *CapturedBIR,
        const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
        BioAPI_BIR_HANDLE *ProcessedBIR);
```

16.32.2 A pair of BIP message types are related to this function: the **process** request BIP message type and the **process** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
Process-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        capturedBIR             BioAPI-INPUT-BIR,
        outputFormat            BioAPI-BIR-BIOMETRIC-DATA-FORMAT OPTIONAL
    }
```

and:

```
Process-ResponseParams ::= SEQUENCE {
        processedBIR            BioAPI-BIR-HANDLE
}
```

16.32.3 When a framework receives a call to the function **BioAPI_Process** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **process** request/response BIP message pair as specified in clause 27, using 16.32.5 and 16.32.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.32.4 When a framework receives (see 13.9) a **process** request BIP message from a master endpoint, it shall process the request as specified via an internal BioAPI function call to **BioAPI_Process** to create and send a corresponding **process** response BIP message in clause 28, using 16.32.5 and 16.32.6 to convert between function parameters and ASN.1 components when required within that clause.

16.32.5 Conversion between the parameters of the C function **BioAPI_Process** and the ASN.1 type **Process-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 78.

Table 78 – Mapping between the parameters of the function BioAPI_Process and the ASN.1 type Process-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
CapturedBIR	capturedBIR	clause 19 in conjunction with 15.46
OutputFormat	outputFormat	clause 19 in conjunction with 15.18
ProcessedBIR	*none*	clause 22

16.32.6 Conversion between the parameters of the C function **BioAPI_Process** and the ASN.1 type **Process-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 79.

Table 79 – Mapping between the parameters of the function BioAPI_Process and the ASN.1 type Process-ResponseParams

Function parameter	Component of the ASN.1 type	References
ProcessedBIR	processedBIR	clause 20 in conjunction with 15.12

16.33 Function BioAPI_ProcessWithAuxBIR

16.33.1 This function is declared in BioAPI as follows:

```
BioAPI_Return BioAPI BioAPI_ProcessWithAuxBIR
        (BioAPI_HANDLE BSPHandle,
        const BioAPI_INPUT_BIR *CapturedBIR,
        const BioAPI_INPUT_BIR *AuxiliaryData,
        const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
        BioAPI_HANDLE *ProcessedBIR);
```

16.33.2 A pair of BIP message types are related to this function: the **processWithAuxBIR** request BIP message type and the **processWithAuxBIR** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
ProcessWithAuxBIR-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        capturedBIR             BioAPI-INPUT-BIR,
        auxiliaryData           BioAPI-INPUT-BIR,
        outputFormat            BioAPI-BIR-BIOMETRIC-DATA-FORMAT OPTIONAL
    }
```

and:

```
ProcessWithAuxBIR-ResponseParams ::= SEQUENCE {
        processedBIR            BioAPI-BIR-HANDLE
    }
```

16.33.3 When a framework receives a call to the function **BioAPI_ProcessWithAuxBIR** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **processWithAuxBIR** request/response BIP message pair as specified in clause 27, using 16.33.5 and 16.33.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.33.4 When a framework receives (see 13.9) a **processWithAuxBIR** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_ProcessWithAuxBIR** to create and send a corresponding **processWithAuxBIR** response BIP message as specified in clause 28, using 16.33.5 and 16.33.6 to convert between function parameters and ASN.1 components when required within that clause.

16.33.5 Conversion between the parameters of the C function **BioAPI_ProcessWithAuxBIR** and the ASN.1 type **ProcessWithAuxBIR-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 80.

Table 80 – Mapping between the parameters of the function BioAPI_ProcessWithAuxBIR and the ASN.1 type ProcessWithAuxBIR-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
CapturedBIR	capturedBIR	clause 19 in conjunction with 15.46
AuxiliaryData	auxiliaryData	clause 19 in conjunction with 15.46
OutputFormat	outputFormat	clause 19 in conjunction with 15.8
ProcessedBIR	*none*	clause 22

16.33.6 Conversion between the parameters of the C function **BioAPI_ProcessWithAuxBIR** and the ASN.1 type **ProcessWithAuxBIR-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 81.

Table 81 – Mapping between the parameters of the function BioAPI_ProcessWithAuxBIR and the ASN.1 type ProcessWithAuxBIR-ResponseParams

Function parameter	Component of the ASN.1 type	References
ProcessedBIR	processedBIR	clause 20 in conjunction with 15.12

16.34 Function BioAPI_VerifyMatch

16.34.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_VerifyMatch
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_FMR MaxFMRRequested,
        const BioAPI_INPUT_BIR *ProcessedBIR,
        const BioAPI_INPUT_BIR *ReferenceTemplate,
        BioAPI_BIR_HANDLE *AdaptedBIR,
        BioAPI_BOOL *Result,
        BioAPI_FMR *FMRAchieved,
        BioAPI_DATA *Payload);
```

16.34.2 A pair of BIP message types are related to this function: the **verifyMatch** request BIP message type and the **verifyMatch** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
VerifyMatch-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        maxFMRRequested         BioAPI-FMR,
        processedBIR            BioAPI-INPUT-BIR,
        referenceTemplate       BioAPI-INPUT-BIR,
        no-adaptedBIR           BOOLEAN,
        no-fmrAchieved          BOOLEAN,
        no-payload              BOOLEAN
    }
```

and:

```
VerifyMatch-ResponseParams ::= SEQUENCE {
        adaptedBIR              BioAPI-BIR-HANDLE OPTIONAL,
        result                  BOOLEAN,
        fmrAchieved             BioAPI-FMR OPTIONAL,
        payload                 BioAPI-DATA OPTIONAL
    }
```

16.34.3 When a framework receives a call to the function **BioAPI_VerifyMatch** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **verifyMatch** request/response BIP message pair as specified in clause 27, using 16.34.5 and 16.34.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.34.4 When a framework receives (see 13.9) a **verifyMatch** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_VerifyMatch** to create and send a corresponding **verifyMatch** response BIP message as specified in clause 27, using 16.34.5 and 16.34.6 to convert between function parameters and ASN.1 components when required within that clause.

16.34.5 Conversion between the parameters of the C function **BioAPI_VerifyMatch** and the ASN.1 type **VerifyMatch-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 82.

Table 82 – Mapping between the parameters of the function BioAPI_VerifyMatch and the ASN.1 type VerifyMatch-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
MaxFMRRequested	maxFMRRequested	15.32
ProcessedBIR	processedBIR	clause 19 in conjunction with 15.46
ReferenceTemplate	referenceTemplate	clause 19 in conjunction with 15.46
AdaptedBIR	no-adaptedBIR	clause 21
Result	*none*	clause 22
FMRAchieved	no-fmrAchieved	clause 21
Payload	no-payload	clause 21

16.34.6 Conversion between the parameters of the C function **BioAPI_VerifyMatch** and the ASN.1 type **VerifyMatch-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 83.

Table 83 – Mapping between the parameters of the function BioAPI_VerifyMatch and the ASN.1 type VerifyMatch-ResponseParams

Function parameter	Component of the ASN.1 type	References
AdaptedBIR	adaptedBIR	clause 20 in conjunction with 15.12
Result	result	clause 20 in conjunction with 15.18
FMRAchieved	fmrAchieved	clause 20 in conjunction with 15.32
Payload	payload	clause 20 in conjunction with 15.22

16.35 Function BioAPI_IdentifyMatch

16.35.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_IdentifyMatch
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_FMR MaxFMRRequested,
        const BioAPI_INPUT_BIR *ProcessedBIR,
        const BioAPI_IDENTIFY_POPULATION  *Population,
        uint32_t TotalNumberOfTemplates,
        BioAPI_BOOL Binning,
        uint32_t MaxNumberOfResults,
        uint32_t *NumberOfResults,
        BioAPI_CANDIDATE **Candidates,
        int32_t Timeout);
```

16.35.2 A pair of BIP message types are related to this function: the **identifyMatch** request BIP message type and the **identifyMatch** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
IdentifyMatch-RequestParams ::= SEQUENCE {
        originalBSPHandle               BioAPI-HANDLE,
        maxFMRRequested                 BioAPI-FMR,
        processedBIR                    BioAPI-INPUT-BIR,
        population                      BioAPI-IDENTIFY-POPULATION,
        totalNumberOfTemplates          UnsignedInt,
        binning                         BOOLEAN,
        maxNumberOfResults              UnsignedInt,
        timeout                         SignedInt
    }
```

and:

```
IdentifyMatch-ResponseParams ::= SEQUENCE {
        candidates                      SEQUENCE    (SIZE(0..max-unsigned-int)) OF
                                            candidate BioAPI-CANDIDATE
    }
```

16.35.3 When a framework receives a call to the function **BioAPI_IdentifyMatch** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint an **identifyMatch** request/response BIP message pair as specified in clause 27, using 16.35.5 and 16.35.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.35.4 When a framework receives (see 13.9) an **identifyMatch** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_IdentifyMatch** to create and send a corresponding **identifyMatch** response BIP message as specified in clause 28, using 16.35.5 and 16.35.6 to convert between function parameters and ASN.1 components when required within that clause.

16.35.5 Conversion between the parameters of the C function **BioAPI_IdentifyMatch** and the ASN.1 type **IdentifyMatch-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 84.

Table 84 – Mapping between the parameters of the function BioAPI_IdentifyMatch and the ASN.1 type IdentifyMatch-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
MaxFMRRequested	maxFMRRequested	15.32
ProcessedBIR	processedBIR	clause 19 in conjunction with 15.46
Population	population	clause 19 in conjunction with 15.43
TotalNumberOfTemplates	totalNumberOfTemplates	15.1.5
Binning	binning	15.18
MaxNumberOfResults	maxNumberOfResults	15.1.5
NumberOfResults	*none*	clause 22
Candidates	*none*	clause 22
Timeout	timeout	15.1.6

16.35.6 Conversion between the parameters of the C function **BioAPI_IdentifyMatch** and the ASN.1 type **IdentifyMatch-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 85.

Table 85 – Mapping between the parameters of the function BioAPI_IdentifyMatch and the ASN.1 type IdentifyMatch-ResponseParams

Function parameter	Component of the ASN.1 type	References
NumberOfResults, Candidates	candidates	clause 20 in conjunction with 16.35.7 and 16.35.8

16.35.7 Conversion from the pair of C variables pointed to by the parameters **Candidates**/**NumberOfResults** to the ASN.1 component **candidates** shall be done as follows: Calling *N* the value of the C variable pointed to by the parameter **NumberOfResults**, each of the first *N* elements (of type **BioAPI_CANDIDATE** – see 15.20) in the array pointed to by the C variable pointed to by the parameter **Candidates** shall be converted, in order, to an element of the component **candidates** as specified in 15.20. The component **candidates** shall have exactly *N* elements.

16.35.8 Conversion from the ASN.1 component **candidates** to the pair of C variables pointed to by the parameters **Candidates**/**NumberOfResults** shall be done as follows: Calling *N* the number of elements of the component **candidates**, a newly allocated array of *N* elements of type **BioAPI_CANDIDATE** (see 15.20) shall be filled by converting each element of the component **candidates**, in order, to an element of the array as specified in 15.20. The C variable pointed to by the parameter **Candidates** shall be set to the address of the array, and the C variable pointed to by the parameter **NumberOfResults** shall be set to *N*.

16.36 Function BioAPI_Enroll

16.36.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Enroll
        (BioAPI_HANDLE BSPHandle,
        BioAPI_BIR_PURPOSE Purpose,
        BioAPI_BIR_SUBTYPE Subtype,
        const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
        const BioAPI_INPUT_BIR *ReferenceTemplate,
        BioAPI_BIR_HANDLE *NewTemplate,
        const BioAPI_DATA *Payload,
        int32_t Timeout,
        BioAPI_BIR_HANDLE *AuditData,
        BioAPI_UUID *TemplateUuid);
```

16.36.2 A pair of BIP message types are related to this function: the **enroll** request BIP message type and the **enroll** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
Enroll-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        purpose                 BioAPI-BIR-PURPOSE,
        subtype                 BioAPI-BIR-SUBTYPE,
        outputFormat            BioAPI-BIR-BIOMETRIC-DATA-FORMAT OPTIONAL,
        referenceTemplate       BioAPI-INPUT-BIR OPTIONAL,
        payload                 BioAPI-DATA OPTIONAL,
        timeout                 SignedInt,
        no-auditData            BOOLEAN,
        no-templateUuid         BOOLEAN
    }
```

and:

```
Enroll-ResponseParams ::= SEQUENCE {
        newTemplate             BioAPI-BIR-HANDLE,
        auditData               BioAPI-BIR-HANDLE OPTIONAL,
        templateUuid        BioAPI-UUID OPTIONAL
    }
```

16.36.3 When a framework receives a call to the function **BioAPI_Enroll** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **enroll** request/response BIP message pair as specified in clause 27, using 16.36.5 and 16.36.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.36.4 When a framework receives (see 13.9) an **enroll** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_Enroll** to create and send a corresponding **enroll** response BIP message as specified in clause 28, using 16.36.5 and 16.36.6 to convert between function parameters and ASN.1 components when required within that clause.

16.36.5 Conversion between the parameters of the C function **BioAPI_Enroll** and the ASN.1 type **Enroll-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 86.

Table 86 – Mapping between the parameters of the function BioAPI_Enroll and the ASN.1 type Enroll-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
Purpose	purpose	15.14
Subtype	subtype	15.16
OutputFormat	outputFormat	clause 19 in conjunction with 15.46
ReferenceTemplate	referenceTemplate	clause 19 in conjunction with 15.46
NewTemplate	*none*	clause 22
Payload	payload	clause 19 in conjunction with 15.22
Timeout	timeout	15.1.6
AuditData	no-auditData	clause 21
TemplateUuid	no-templateUuid	clause 21

16.36.6 Conversion between the parameters of the C function **BioAPI_Enroll** and the ASN.1 type **Enroll-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 87.

Table 87 – Mapping between the parameters of the function BioAPI_Enroll and the ASN.1 type Enroll-ResponseParams

Function parameter	Component of the ASN.1 type	References
NewTemplate	newTemplate	clause 20 in conjunction with 15.12
AuditData	auditData	clause 20 in conjunction with 15.12
TemplateUuid	templateUuid	clause 20 in conjunction with 15.58

16.37 Function BioAPI_Verify

16.37.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Verify
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_FMR MaxFMRRequested,
        const BioAPI_INPUT_BIR *ReferenceTemplate,
        BioAPI_BIR_SUBTYPE Subtype,
        BioAPI_BIR_HANDLE *AdaptedBIR,
        BioAPI_BOOL *Result,
        BioAPI_FMR *FMRAchieved,
        BioAPI_DATA *Payload,
        int32_t Timeout,
        BioAPI_BIR_HANDLE *AuditData);
```

16.37.2 A pair of BIP message types are related to this function: the **verify** request BIP message type and the **verify** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
Verify-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        maxFMRRequested         BioAPI-FMR,
        referenceTemplate       BioAPI-INPUT-BIR,
        subtype                 BioAPI-BIR-SUBTYPE,
        timeout                 SignedInt,
        no-adaptedBIR           BOOLEAN,
        no-fmrAchieved          BOOLEAN,
        no-payload              BOOLEAN,
        no-auditData            BOOLEAN
    }
```

and:

```
Verify-ResponseParams ::= SEQUENCE {
        adaptedBIR          BioAPI-BIR-HANDLE OPTIONAL,
        result              BOOLEAN,
        fmrAchieved         BioAPI-FMR OPTIONAL,
        payload             BioAPI-DATA OPTIONAL,
        auditData           BioAPI-BIR-HANDLE OPTIONAL
    }
```

16.37.3 When a framework receives a call to the function **BioAPI_Verify** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **verify** request/response BIP message pair as specified in clause 27, using 16.37.5 and 16.37.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.37.4 When a framework receives (see 13.9) a **verify** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_Verify** to create and send a corresponding **verify** response BIP message as specified in clause 28, using 16.37.5 and 16.37.6 to convert between function parameters and ASN.1 components when required within that clause.

16.37.5 Conversion between the parameters of the C function **BioAPI_Verify** and the ASN.1 type **Verify-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 88.

Table 88 – Mapping between the parameters of the function BioAPI_Verify and the ASN.1 type Verify-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
MaxFMRRequested	maxFMRRequested	15.32
ReferenceTemplate	referenceTemplate	clause 19 in conjunction with 15.46
Subtype	subtype	15.16
AdaptedBIR	no-adaptedBIR	clause 21
Result	*none*	clause 22
FMRAchieved	no-fmrAchieved	clause 21
Payload	no-payload	clause 21
Timeout	timeout	15.1.6
AuditData	no-auditData	clause 21

16.37.6 Conversion between the parameters of the C function **BioAPI_Verify** and the ASN.1 type **Verify-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 89.

Table 89 – Mapping between the parameters of the function BioAPI_Verify and the ASN.1 type Verify-ResponseParams

Function parameter	Component of the ASN.1 type	References
AdaptedBIR	adaptedBIR	clause 20 in conjunction with 15.12
Result	result	clause 20 in conjunction with 15.18
FMRAchieved	fmrAchieved	clause 20 in conjunction with 15.32
Payload	payload	clause 20 in conjunction with 15.22
AuditData	auditData	clause 20 in conjunction with 15.12

16.38 Function BioAPI_Identify

16.38.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Identify
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_FMR MaxFMRRequested,
        BioAPI_BIR_SUBTYPE Subtype,
        const BioAPI_IDENTIFY_POPULATION *Population,
        uint32_t  TotalNumberOfTemplates,
        BioAPI_BOOL  Binning,
        uint32_t  MaxNumberOfResults,
        uint32_t  *NumberOfResults,
        BioAPI_CANDIDATE **Candidates,
        int32_t  Timeout,
        BioAPI_BIR_HANDLE *AuditData);
```

16.38.2 A pair of BIP message types are related to this function: the **identify** request BIP message type and the **identify** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
Identify-RequestParams ::= SEQUENCE {
        originalBSPHandle               BioAPI-HANDLE,
        maxFMRRequested                 BioAPI-FMR,
        subtype                         BioAPI-BIR-SUBTYPE,
        population                      BioAPI-IDENTIFY-POPULATION,
        totalNumberOfTemplates          UnsignedInt,
        binning                         BOOLEAN,
        maxNumberOfResults              UnsignedInt,
        timeout                         SignedInt,
        no-auditData                    BOOLEAN
    }
```

and:

```
Identify-ResponseParams ::= SEQUENCE {
        candidates                      SEQUENCE(SIZE(0..max-unsigned-int)) OF
                                            candidate BioAPI-CANDIDATE,
        auditData                       BioAPI-BIR-HANDLE OPTIONAL
    }
```

16.38.3 When a framework receives a call to the function **BioAPI_Identify** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint an **identify** request/response BIP message pair as specified in clause 27, using 16.38.5 and 16.38.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.38.4 When a framework receives (see 13.9) an **identify** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_Identify** to create and send a corresponding **identify** response BIP message as specified in clause 28, using 16.38.5 and 16.38.6 to convert between function parameters and ASN.1 components when required within that clause.

16.38.5 Conversion between the parameters of the C function **BioAPI_Identify** and the ASN.1 type **Identify-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 90.

Table 90 – Mapping between the parameters of the function BioAPI_Identify and the ASN.1 type Identify-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
MaxFMRRequested	maxFMRRequested	15.32
Subtype	subtype	15.16
Population	population	clause 19 in conjunction with 15.43
TotalNumberOfTemplates	totalNumberOfTemplates	15.1.5
Binning	binning	15.18
MaxNumberOfResults	maxNumberOfResults	15.1.5
NumberOfResults	*none*	clause 22
Candidates	*none*	clause 22
Timeout	timeout	15.1.6
AuditData	no-auditData	clause 21

16.38.6 Conversion between the parameters of the C function **BioAPI_Identify** and the ASN.1 type **Identify-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 91.

Table 91 – Mapping between the parameters of the function BioAPI_Identify and the ASN.1 type Identify-ResponseParams

Function parameter	Component of the ASN.1 type	References
NumberOfResults, Candidates	candidates	clause 20 in conjunction with 16.38.7 and 16.38.8
AuditData	auditData	clause 20 in conjunction with 15.12

16.38.7 Conversion from the pair of C variables pointed to by the parameters **Candidates**/**NumberOfResults** to the ASN.1 component **candidates** shall be done as follows: Calling *N* the value of the C variable pointed to by the parameter **NumberOfResults**, each of the first *N* elements (of type **BioAPI_CANDIDATE** – see 15.20) in the array pointed to by the C variable pointed to by the parameter **Candidates** shall be converted, in order, to an element of the component **candidates** as specified in 15.20. The component **candidates** shall have exactly *N* elements.

16.38.8 Conversion from the ASN.1 component **candidates** to the pair of C variables pointed to by the parameters **Candidates**/**NumberOfResults** shall be done as follows: Calling *N* the number of elements of the component **candidates**, a newly allocated array of *N* elements of type **BioAPI_CANDIDATE** (see 15.20) shall be filled by converting each element of the component **candidates**, in order, to an element of the array as specified in 15.20. The C variable pointed to by the parameter **Candidates** shall be set to the address of the array, and the C variable pointed to by the parameter **NumberOfResults** shall be set to *N*.

16.39 Function BioAPI_Import

16.39.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Import
        (BioAPI_HANDLE  BSPHandle,
        const BioAPI_DATA *InputData,
        const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *InputFormat,
        const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat,
        BioAPI_BIR_PURPOSE Purpose,
        BioAPI_BIR_HANDLE *ConstructedBIR);
```

16.39.2 A pair of BIP message types are related to this function: the **import** request BIP message type and the **import** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
Import-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        inputData               BioAPI-DATA,
        inputFormat             BioAPI-BIR-BIOMETRIC-DATA-FORMAT,
        outputFormat            BioAPI-BIR-BIOMETRIC-DATA-FORMAT OPTIONAL,
        purpose                 BioAPI-BIR-PURPOSE
    }
```

and:

```
Import-ResponseParams ::= SEQUENCE {
        constructedBIR          BioAPI-BIR-HANDLE
    }
```

16.39.3 When a framework receives a call to the function **BioAPI_Import** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint an **import** request/response BIP message pair as specified in clause 27, using 16.39.5 and 16.39.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.39.4 When a framework receives (see 13.9) an **import** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_Import** to create and send a corresponding **import** response BIP message as specified in clause 28, using 16.39.5 and 16.39.6 to convert between function parameters and ASN.1 components when required within that clause.

16.39.5 Conversion between the parameters of the C function **BioAPI_Import** and the ASN.1 type **Import-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 92.

Table 92 – Mapping between the parameters of the function BioAPI_Import and the ASN.1 type Import-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
InputData	inputData	clause 19 in conjunction with 15.22
InputFormat	inputFormat	clause 19 in conjunction with 15.8
OutputFormat	outputFormat	clause 19 in conjunction with 15.8
Purpose	purpose	15.14
ConstructedBIR	*none*	clause 22

16.39.6 Conversion between the parameters of the C function **BioAPI_Import** and the ASN.1 type **Import-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 93.

Table 93 – Mapping between the parameters of the function BioAPI_Import and the ASN.1 type Import-ResponseParams

Function parameter	Component of the ASN.1 type	References
ConstructedBIR	constructedBIR	clause 20 in conjunction with 15.12

16.40 Function BioAPI_PresetIdentifyPopulation

16.40.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_PresetIdentifyPopulation
        (BioAPI_HANDLE  BSPHandle,
        const BioAPI_IDENTIFY_POPULATION  *Population);
```

16.40.2 A pair of BIP message types are related to this function: the **presetIdentifyPopulation** request BIP message type and the **presetIdentifyPopulation** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
PresetIdentifyPopulation-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        population              BioAPI-IDENTIFY-POPULATION
    }
```

and:

```
PresetIdentifyPopulation-ResponseParams ::= NULL
```

16.40.3 When a framework receives a call to the function **BioAPI_PresetIdentifyPopulation** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **presetIdentifyPopulation** request/response BIP message pair as specified in clause 27, using 16.40.5 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.40.4 When a framework receives (see 13.9) a **presetIdentifyPopulation** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_PresetIdentifyPopulation** to create and send a corresponding **presetIdentifyPopulation** response BIP message as specified in clause 28, using 16.40.5 to convert between function parameters and ASN.1 components when required within that clause.

16.40.5 Conversion between the parameters of the C function **BioAPI_PresetIdentifyPopulation** and the ASN.1 type **PresetIdentifyPopulation-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 94.

Table 94 – Mapping between the parameters of the function BioAPI_PresetIdentifyPopulation and the ASN.1 type PresetIdentifyPopulation-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
Population	population	clause 19 in conjunction with 15.43

16.41 Function BioAPI_Transform

16.41.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Transform
        (BioAPI_HANDLE BSPHandle,
        const BioAPI_UUID *OperationUuid,
        const BioAPI_INPUT_BIR *InputBIRs,
        uint32_t NumberOfInputBIRs,
        BioAPI_BIR_HANDLE **OutputBIRs,
        uint32_t *NumberOfOutputBIRs)
```

16.41.2 A pair of BIP message types are related to this function: the **transform** request BIP message type and the **transform** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
Transform-RequestParams ::= SEQUENCE {
        bspHandle               BioAPI-HANDLE,
        operationUuid           BioAPI-UUID,
        inputBIRs               SEQUENCE (SIZE(0..max-unsigned-int)) OF
                                        BioAPI-INPUT-BIR
    }
```

and:

```
Transform-ResponseParams ::= SEQUENCE {
        outputBIRs              SEQUENCE    (SIZE(0..max-unsigned-int)) OF
                                        BioAPI-BIR-HANDLE
    }
```

16.41.3 When a framework receives a call to the function **BioAPI_Transform** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 26. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a transform request/response BIP message pair as specified in clause 27, using 16.41.5 and 16.41.8 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.41.4 When a framework receives (see 13.9) a transform request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_Transform** to create and send a corresponding transform response BIP message as specified in clause 28, using 16.41.5 and 16.41.8 to convert between function parameters and ASN.1 components when required within that clause.

16.41.5 Conversion between the parameters of the C function **BioAPI_Transform** and the ASN.1 type **Transform-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 95.

Table 95 – Mapping between the parameters of the function BioAPI_Transform and the ASN.1 type Transform-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
OperationUuid	operationUuid	15.58
InputBIRs, NumberOfInputBIRs	inputBIRs	16.41.6 and 16.41.7
OutputBIRs	*none*	clause 22
NumberOfOutputBIRs	*none*	clause 22

16.41.6 Conversion from the pair of parameters **InputBIRs**/**NumberOfInputBIRs** to the ASN.1 component **inputBIRs** shall be done as follows: Calling *N* the value of the parameter **NumberOfInputBIRs**, each of the first N elements (of type **BioAPI_INPUT_BIR** – see 15.46) in the array pointed to by the parameter **InputBIRs** shall be converted, in order, to an element of the component **inputBIRs** as specified in 15.46.3. The component **inputBIRs** shall have exactly *N* elements.

16.41.7 Conversion from the ASN.1 component **inputBIRs** to the pair of parameters **InputBIRs**/**NumberOfInputBIRs** shall be done as follows: Calling *N* the number of elements of the component **guiBitmaps**, a newly allocated array of *N* elements of type **BioAPI_INPUT_BIR** (see 15.46) shall be filled by converting each element of the component **inputBIRs**, in order, to an element of the array as specified in 15.46.4. The parameter **InputBIRs** shall be set to the address of the array, and the parameter **NumberOfInputBIRs** shall be set to *N*.

16.41.8 Conversion between the parameters of the C function **BioAPI_Transform** and the ASN.1 type **Transform-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 96.

Table 96 – Mapping between the parameters of the function BioAPI_Transfrom and the ASN.1 type Transform-ResponseParams

Function parameter	Component of the ASN.1 type	References
OutputBIRs, NumberOfOuputBIRs	ouputBIRs	clause 29 in conjunction with 16.41.9 and 16.41.10

16.41.9 Conversion from the pair of C variables pointed to by the parameters **OutputBIRs**/**NumberOfOutputBIRs** to the ASN.1 component **outputBIRs** shall be done as follows: Calling *N* the value of the C variable pointed to by the parameter **NumberOfOutputBIRs**, each of the first *N* elements (of type **BioAPI_BIR_HANDLE** – see 15.12) in the array pointed to by the C variable pointed to by the parameter **OutputBIRs** shall be converted, in order, to an element of the component **outputBIRs** as specified in 15.46.3. The component **outputBIRs** shall have exactly *N* elements.

16.41.10 Conversion from the ASN.1 component **outputBIRs** to the pair of C variables pointed to by the parameters **OutputBIRs**/**NumberOfOutputBIRs** shall be done as follows: Calling *N* the number of elements of the component **outputBIRs**, a newly allocated array of *N* elements of type **BioAPI_BIR_HANDLE** (see 15.12) shall be filled by converting each element of the component **outputBIRs**, in order, to an element of the array as specified in 15.46.4. The C variable pointed to by the parameter **OutputBIRs** shall be set to the address of the array, and the C variable pointed to by the parameter **NumberOfOutputBIRs** shall be set to *N*.

16.42 Function BioAPI_DbOpen

16.42.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_DbOpen
        (BioAPI_HANDLE  BSPHandle,
        const BioAPI_UUID *DbUuid,
        BioAPI_DB_ACCESS_TYPE AccessRequest,
        BioAPI_DB_HANDLE *DbHandle,
        BioAPI_DB_MARKER_HANDLE *MarkerHandle);
```

16.42.2 A pair of BIP message types are related to this function: the **dbOpen** request BIP message type and the **dbOpen** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
DbOpen-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        dbUuid                  BioAPI-UUID,
        accessRequest           BioAPI-DB-ACCESS-TYPE
    }
```

and:

```
DbOpen-ResponseParams ::= SEQUENCE {
        dbHandle                BioAPI-DB-HANDLE,
        markerHandle            BioAPI-DB-MARKER-HANDLE
    }
```

16.42.3 When a framework receives a call to the function **BioAPI_DbOpen** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **dbOpen** request/response BIP message pair as specified in clause 27, using 16.42.5 and 16.42.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.42.4 When a framework receives (see 13.9) a **dbOpen** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_DbOpen** to create and send a corresponding **dbOpen** response BIP message as specified in clause 28, using 16.42.5 and 16.42.6 to convert between function parameters and ASN.1 components when required within that clause.

16.42.5 Conversion between the parameters of the C function **BioAPI_DbOpen** and the ASN.1 type **DbOpen-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 97.

Table 97 – Mapping between the parameters of the function BioAPI_DbOpen and the ASN.1 type DbOpen-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
DbUuid	dbUuid	clause 19 in conjunction with 15.58
AccessRequest	accessRequest	15.24
DbHandle	*none*	clause 22
MarkerHandle	*none*	clause 22

16.42.6 Conversion between the parameters of the C function **BioAPI_DbOpen** and the ASN.1 type **DbOpen-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 98.

Table 98 – Mapping between the parameters of the function BioAPI_DbOpen and the ASN.1 type DbOpen-ResponseParams

Function parameter	Component of the ASN.1 type	References
DbHandle	dbHandle	clause 20 in conjunction with 15.26
MarkerHandle	marker	clause 20 in conjunction with 15.25

16.43 Function BioAPI_DbClose

16.43.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_DbClose
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_DB_HANDLE DbHandle);
```

16.43.2 A pair of BIP message types are related to this function: the **dbClose** request BIP message type and the **dbClose** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
DbClose-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        dbHandle                BioAPI-DB-HANDLE
}
```

and:

```
DbClose-ResponseParams ::= NULL
```

16.43.3 When a framework receives a call to the function **BioAPI_DbClose** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **dbClose** request/response BIP message pair as specified in clause 27, using 16.43.5 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.43.4 When a framework receives (see 13.9) a **dbClose** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_DbClose** to create and send a corresponding **dbClose** response BIP message as specified in clause 28, using 16.43.5 to convert between function parameters and ASN.1 components when required within that clause.

16.43.5 Conversion between the parameters of the C function **BioAPI_DbClose** and the ASN.1 type **DbClose-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 99.

Table 99 – Mapping between the parameters of the function BioAPI_DbClose and the ASN.1 type DbClose-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
DbHandle	dbHandle	15.26

16.44 Function BioAPI_DbCreate

16.44.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_DbCreate
        (BioAPI_HANDLE  BSPHandle,
        const BioAPI_UUID *DbUuid,
        uint32_t NumberOfRecords,
        BioAPI_DB_ACCESS_TYPE  AccessRequest,
        BioAPI_DB_HANDLE *DbHandle);
```

16.44.2 A pair of BIP message types are related to this function: the **dbCreate** request BIP message type and the **dbCreate** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
DbCreate-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        dbUuid                  BioAPI-UUID,
        numberOfRecords         UnsignedInt,
        accessRequest           BioAPI-DB-ACCESS-TYPE
    }
```

and:

```
DbCreate-ResponseParams ::= SEQUENCE {
        dbHandle                BioAPI-DB-HANDLE
    }
```

16.44.3 When a framework receives a call to the function **BioAPI_DbCreate** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **dbCreate** request/response BIP message pair as specified in clause 27, using 16.44.5 and 16.44.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.44.4 When a framework receives (see 13.9) a **dbCreate** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_DbCreate** to create and send a corresponding **dbCreate** response BIP message as specified in clause 28, using 16.44.5 and 16.44.6 to convert between function parameters and ASN.1 components when required within that clause.

16.44.5 Conversion between the parameters of the C function **BioAPI_DbCreate** and the ASN.1 type **DbCreate-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 100.

Table 100 – Mapping between the parameters of the function BioAPI_DbCreate and the ASN.1 type DbCreate-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
DbUuid	dbUuid	clause 19 in conjunction with 15.58
NumberOfRecords	numberOfRecords	15.1.5
AccessRequest	accessRequest	15.24
DbHandle	*none*	clause 22

16.44.6 Conversion between the parameters of the C function **BioAPI_DbCreate** and the ASN.1 type **DbCreate-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 101.

Table 101 – Mapping between the parameters of the function BioAPI_DbCreate and the ASN.1 type DbCreate-ResponseParams

Function parameter	Component of the ASN.1 type	References
DbHandle	dbHandle	clause 20 in conjunction with 15.26

16.45 Function BioAPI_DbDelete

16.45.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_DbDelete
        (BioAPI_HANDLE BSPHandle,
        const BioAPI_UUID *DbUuid);
```

16.45.2 A pair of BIP message types are related to this function: the **dbDelete** request BIP message type and the **dbDelete** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
DbDelete-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        dbUuid                  BioAPI-UUID
    }
```

and:

```
DbDelete-ResponseParams ::= NULL
```

16.45.3 When a framework receives a call to the function **BioAPI_DbDelete** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **dbDelete** request/response BIP message pair as specified in clause 27, using 16.45.5 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.45.4 When a framework receives (see 13.9) a **dbDelete** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_DbDelete** to create and send a corresponding **dbDelete** response BIP message as specified in clause 28, using 16.45.5 to convert between function parameters and ASN.1 components when required within that clause.

16.45.5 Conversion between the parameters of the C function **BioAPI_DbDelete** and the ASN.1 type **DbDelete-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 102.

Table 102 – Mapping between the parameters of the function BioAPI_DbDelete and the ASN.1 type DbDelete-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
DbUuid	dbUuid	clause 19 in conjunction with 15.58

16.46 Function BioAPI_DbSetMarker

16.46.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_DbSetMarker
        (BioAPI_HANDLE BSPHandle,
        BioAPI_DB_HANDLE DbHandle,
        const BioAPI_UUID *KeyValue,
        BioAPI_DB_MARKER_HANDLE MarkerHandle);
```

16.46.2 A pair of BIP message types are related to this function: the **dbSetMarker** request BIP message type and the **dbSetMarker** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
DbSetMarker-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        dbHandle                BioAPI-DB-HANDLE,
        keyValue                BioAPI-UUID,
        markerHandle            BioAPI-DB-MARKER-HANDLE
    }
```

and:

```
DbSetMarker-ResponseParams ::= NULL
```

16.46.3 When a framework receives a call to the function **BioAPI_DbSetMarker** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **dbSetMarker** request/response BIP message pair as specified in clause 27, using 16.46.5 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.46.4 When a framework receives (see 13.9) a **dbSetMarker** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_DbSetMarker** to create and send a corresponding **dbSetMarker** response BIP message as specified in clause 28, using 16.46.5 to convert between function parameters and ASN.1 components when required within that clause.

16.46.5 Conversion between the parameters of the C function **BioAPI_DbSetMarker** and the ASN.1 type **DbSetMarker-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 103.

Table 103 – Mapping between the parameters of the function BioAPI_DbSetMarker and the ASN.1 type DbSetMarker-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
DbHandle	dbHandle	15.26
KeyValue	keyValue	clause 19 in conjunction with 15.58
MarkerHandle	markerHandle	15.25

16.47 Function BioAPI_DbFreeMarker

16.47.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_DbFreeMarker
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_DB_MARKER_HANDLE MarkerHandle);
```

16.47.2 A pair of BIP message types are related to this function: the **dbFreeMarker** request BIP message type and the **dbFreeMarker** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
DbFreeMarker-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        markerHandle            BioAPI-DB-MARKER-HANDLE
    }
```

and:

```
DbFreeMarker-ResponseParams ::= NULL
```

16.47.3 When a framework receives a call to the function **BioAPI_DbFreeMarker** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **dbFreeMarker** request/response BIP message pair as specified in clause 27, using 16.47.5 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.47.4 When a framework receives (see 13.9) a **dbFreeMarker** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_DbFreeMarker** to create and send a corresponding **dbFreeMarker** response BIP message as specified in clause 28, using 16.47.5 to convert between function parameters and ASN.1 components when required within that clause.

16.47.5 Conversion between the parameters of the C function **BioAPI_DbFreeMarker** and the ASN.1 type **DbFreeMarker-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 104.

Table 104 – Mapping between the parameters of the function BioAPI_DbFreeMarker and the ASN.1 type DbFreeMarker-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
MarkerHandle	markerHandle	15.25

16.48 Function BioAPI_DbStoreBIR

16.48.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_DbStoreBIR
        (BioAPI_HANDLE BSPHandle,
        const BioAPI_INPUT_BIR *BIRToStore,
        BioAPI_DB_HANDLE DbHandle,
        BioAPI_UUID *BirUuid);
```

16.48.2 A pair of BIP message types are related to this function: the **dbStoreBIR** request BIP message type and the **dbStoreBIR** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
DbStoreBIR-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        birToStore              BioAPI-INPUT-BIR,
        dbHandle                BioAPI-DB-HANDLE
    }
```

and:

```
DbStoreBIR-ResponseParams ::= SEQUENCE {
        birUuid                 BioAPI-UUID
    }
```

16.48.3 When a framework receives a call to the function **BioAPI_DbStoreBIR** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter

BSPHandle as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **dbStoreBIR** request/response BIP message pair as specified in clause 27, using 16.48.5 and 16.48.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.48.4 When a framework receives (see 13.9) a **dbStoreBIR** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_DbStoreBIR** to create and send a corresponding **dbStoreBIR** response BIP message as specified in clause 28, using 16.48.5 and 16.48.6 to convert between function parameters and ASN.1 components when required within that clause.

16.48.5 Conversion between the parameters of the C function **BioAPI_DbStoreBIR** and the ASN.1 type **DbStoreBIR-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 105.

Table 105 – Mapping between the parameters of the function BioAPI_DbStoreBIR and the ASN.1 type DbStoreBIR-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
BIRToStore	birToStore	clause 19 in conjunction with 15.46
DbHandle	dbHandle	15.26
BirUuid	*none*	clause 22

16.48.6 Conversion between the parameters of the C function **BioAPI_DbStoreBIR** and the ASN.1 type **DbStoreBIR-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 106.

Table 106 – Mapping between the parameters of the function BioAPI_DbStoreBIR and the ASN.1 type DbStoreBIR-ResponseParams

Function parameter	Component of the ASN.1 type	References
BirUuid	birUuid	clause 20 in conjunction with 15.58

16.49 Function BioAPI_DbGetBIR

16.49.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_DbGetBIR
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_DB_HANDLE  DbHandle,
        const BioAPI_UUID *KeyValue,
        BioAPI_BIR_HANDLE *RetrievedBIR,
        BioAPI_DB_MARKER_HANDLE *MarkerHandle);
```

16.49.2 A pair of BIP message types are related to this function: the **dbGetBIR** request BIP message type and the **dbGetBIR** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
DbGetBIR-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        dbHandle                BioAPI-DB-HANDLE,
        keyValue                BioAPI-UUID
    }
```

and:

```
DbGetBIR-ResponseParams ::= SEQUENCE {
        retrievedBIR            BioAPI-BIR-HANDLE,
        markerHandle            BioAPI-DB-MARKER-HANDLE
    }
```

16.49.3 When a framework receives a call to the function **BioAPI_DbGetBIR** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **dbGetBIR** request/response BIP message pair as specified in clause 27, using 16.49.5 and 16.49.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.49.4 When a framework receives (see 13.9) a **dbGetBIR** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_DbGetBIR** to create and send a corresponding **dbGetBIR** response BIP message as specified in clause 28, using 16.49.5 and 16.49.6 to convert between function parameters and ASN.1 components when required within that clause.

16.49.5 Conversion between the parameters of the C function **BioAPI_DbGetBIR** and the ASN.1 type **DbGetBIR-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 107.

Table 107 – Mapping between the parameters of the function BioAPI_DbGetBIR and the ASN.1 type DbGetBIR-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
DbHandle	dbHandle	15.26
KeyValue	keyValue	clause 19 in conjunction with 15.58
RetrievedBIR	*none*	clause 22
MarkerHandle	*none*	clause 22

16.49.6 Conversion between the parameters of the C function **BioAPI_DbGetBIR** and the ASN.1 type **DbGetBIR-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 108.

Table 108 – Mapping between the parameters of the function BioAPI_DbGetBIR and the ASN.1 type DbGetBIR-ResponseParams

Function parameter	Component of the ASN.1 type	References
RetrievedBIR	retrievedBIR	clause 20 in conjunction with 15.12
MarkerHandle	markerHandle	clause 20 in conjunction with 15.25

16.50 Function BioAPI_DbGetNextBIR

16.50.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_DbGetNextBIR
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_DB_HANDLE DbHandle,
        BioAPI_DB_MARKER_HANDLE MarkerHandle,
        BioAPI_BIR_HANDLE *RetrievedBIR,
        BioAPI_UUID *BirUuid);
```

16.50.2 A pair of BIP message types are related to this function: the **dbGetNextBIR** request BIP message type and the **dbGetNextBIR** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
DbGetNextBIR-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        dbHandle                BioAPI-DB-HANDLE,
        markerHandle            BioAPI-DB-MARKER-HANDLE
}
```

and:

```
DbGetNextBIR-ResponseParams ::= SEQUENCE {
        retrievedBIR            BioAPI-BIR-HANDLE,
        birUuid                 BioAPI-UUID
    }
```

16.50.3 When a framework receives a call to the function **BioAPI_DbGetNextBIR** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **dbGetNextBIR** request/response BIP message pair as specified in clause 27, using 16.50.5 and 16.50.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.50.4 When a framework receives (see 13.9) a **dbGetNextBIR** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_DbGetNextBIR** to create and send a corresponding **dbGetNextBIR** response BIP message as specified in clause 28, using 16.50.5 and 16.50.6 to convert between function parameters and ASN.1 components when required within that clause.

16.50.5 Conversion between the parameters of the C function **BioAPI_DbGetNextBIR** and the ASN.1 type **DbGetNextBIR-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 109.

Table 109 – Mapping between the parameters of the function BioAPI_DbGetNextBIR and the ASN.1 type DbGetNextBIR-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
DbHandle	dbHandle	15.26
MarkerHandle	markerHandle	15.25
RetrievedBIR	*none*	clause 22
BirUuid	*none*	clause 22

16.50.6 Conversion between the parameters of the C function **BioAPI_DbGetNextBIR** and the ASN.1 type **DbGetNextBIR-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 110.

Table 110 – Mapping between the parameters of the function BioAPI_DbGetNextBIR and the ASN.1 type DbGetNextBIR-ResponseParams

Function parameter	Component of the ASN.1 type	References
RetrievedBIR	retrievedBIR	clause 20 in conjunction with 15.12
BirUuid	birUuid	clause 20 in conjunction with 15.58

16.51 Function BioAPI_DbDeleteBIR

16.51.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_DbDeleteBIR
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_DB_HANDLE  DbHandle,
        const BioAPI_UUID *KeyValue);
```

16.51.2 A pair of BIP message types are related to this function: the **dbDeleteBIR** request BIP message type and the **dbDeleteBIR** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
DbDeleteBIR-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        dbHandle                BioAPI-DB-HANDLE,
        keyValue                BioAPI-UUID
    }
```

and:

```
DbDeleteBIR-ResponseParams ::= NULL
```

16.51.3 When a framework receives a call to the function **BioAPI_DbDeleteBIR** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **dbDeleteBIR** request/response BIP message pair as specified in clause 27, using 16.51.5 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.51.4 When a framework receives (see 13.9) a **dbDeleteBIR** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_DbDeleteBIR** to create and send a corresponding **dbDeleteBIR** response BIP message as specified in clause 28, using 16.51.5 to convert between function parameters and ASN.1 components when required within that clause.

16.51.5 Conversion between the parameters of the C function **BioAPI_DbDeleteBIR** and the ASN.1 type **DbDeleteBIR-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 111.

Table 111 – Mapping between the parameters of the function BioAPI_DbDeleteBIR and the ASN.1 type DbDeleteBIR-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
DbHandle	dbHandle	15.26
KeyValue	keyValue	clause 19 in conjunction with 15.58

16.52 Function BioAPI_CalibrateSensor

16.52.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_CalibrateSensor
    (BioAPI_HANDLE  BSPHandle,
    int32_t Timeout);
```

16.52.2 A pair of BIP message types are related to this function: the **calibrateSensor** request BIP message type and the **calibrateSensor** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
CalibrateSensor-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        timeout                 SignedInt
    }
```

and:

```
CalibrateSensor-ResponseParams ::= NULL
```

16.52.3 When a framework receives a call to the function **BioAPI_CalibrateSensor** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **calibrateSensor** request/response BIP message pair as specified in clause 27, using 16.52.5 to convert between function parameters and

ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.52.4 When a framework receives (see 13.9) a **calibrateSensor** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_CalibrateSensor** to create and send a corresponding **calibrateSensor** response BIP message as specified in clause 28, using 16.52.5 to convert between function parameters and ASN.1 components when required within that clause.

16.52.5 Conversion between the parameters of the C function **BioAPI_CalibrateSensor** and the ASN.1 type **CalibrateSensor-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 112.

Table 112 – Mapping between the parameters of the function BioAPI_CalibrateSensor and the ASN.1 type CalibrateSensor-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
Timeout	timeout	15.1.6

16.53 Function BioAPI_SetPowerMode

16.53.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_SetPowerMode
        (BioAPI_HANDLE BSPHandle,
        BioAPI_UNIT_ID UnitID,
        BioAPI_POWER_MODE  PowerMode);
```

16.53.2 A pair of BIP message types are related to this function: the **setPowerMode** request BIP message type and the **setPowerMode** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
SetPowerMode-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        unitID                  BioAPI-UNIT-ID,
        powerMode               BioAPI-POWER-MODE
    }
```

and:

```
SetPowerMode-ResponseParams ::= NULL
```

16.53.3 When a framework receives a call to the function **BioAPI_SetPowerMode** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **setPowerMode** request/response BIP message pair as specified in clause 27, using 16.53.5 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.53.4 When a framework receives (see 13.9) a **setPowerMode** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_SetPowerMode** to create and send a corresponding **setPowerMode** response BIP message as specified in clause 28, using 16.53.5 to convert between function parameters and ASN.1 components when required within that clause.

16.53.5 Conversion between the parameters of the C function **BioAPI_SetPowerMode** and the ASN.1 type **SetPowerMode-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 113.

Table 113 – Mapping between the parameters of the function BioAPI_SetPowerMode and the ASN.1 type SetPowerMode-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
UnitID	unitID	15.55
PowerMode	powerMode	15.50

16.54 Function BioAPI_SetIndicatorStatus

16.54.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_SetIndicatorStatus
        (BioAPI_HANDLE  BSPHandle,
        BioAPI_UNIT_ID UnitID,
        BioAPI_INDICATOR_STATUS IndicatorStatus);
```

16.54.2 A pair of BIP message types are related to this function: the **setIndicatorStatus** request BIP message type and the **setIndicatorStatus** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
SetIndicatorStatus-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        unitID                  BioAPI-UNIT-ID,
        indicatorStatus         BioAPI-INDICATOR-STATUS
    }
```

and:

```
SetIndicatorStatus-ResponseParams ::= NULL
```

16.54.3 When a framework receives a call to the function **BioAPI_SetIndicatorStatus** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **setIndicatorStatus** request/response BIP message pair as specified in clause 27, using 16.54.5 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.54.4 When a framework receives (see 13.9) a **setIndicatorStatus** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_SetIndicatorStatus** to create and send a corresponding **setIndicatorStatus** response BIP message as specified in clause 28, using 16.54.5 to convert between function parameters and ASN.1 components when required within that clause.

16.54.5 Conversion between the parameters of the C function **BioAPI_SetIndicatorStatus** and the ASN.1 type **SetIndicatorStatus-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 114.

Table 114 – Mapping between the parameters of the function BioAPI_SetIndicatorStatus and the ASN.1 type SetIndicatorStatus-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
UnitID	unitID	15.55
IndicatorStatus	indicatorStatus	15.45

16.55 Function BioAPI_GetIndicatorStatus

16.55.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_GetIndicatorStatus
        (BioAPI_HANDLE BSPHandle,
        BioAPI_UNIT_ID UnitID,
        BioAPI_INDICATOR_STATUS *IndicatorStatus);
```

16.55.2 A pair of BIP message types are related to this function: the **getIndicatorStatus** request BIP message type and the **getIndicatorStatus** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
GetIndicatorStatus-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE,
        unitID                  BioAPI-UNIT-ID
    }
```

and:

```
GetIndicatorStatus-ResponseParams ::= SEQUENCE {
        indicatorStatus         BioAPI-INDICATOR-STATUS
    }
```

16.55.3 When a framework receives a call to the function **BioAPI_GetIndicatorStatus** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **getIndicatorStatus** request/response BIP message pair as specified in clause 27, using 16.55.5 and 16.55.6 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.55.4 When a framework receives (see 13.9) a **getIndicatorStatus** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_GetIndicatorStatus** to create and send a corresponding **getIndicatorStatus** response BIP message as specified in clause 28, using 16.55.5 and 16.55.6 to convert between function parameters and ASN.1 components when required within that clause.

16.55.5 Conversion between the parameters of the C function **BioAPI_GetIndicatorStatus** and the ASN.1 type **GetIndicatorStatus-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 115.

Table 115 – Mapping between the parameters of the function BioAPI_GetIndicatorStatus and the ASN.1 type GetIndicatorStatus-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26
UnitID	unitID	15.55
IndicatorStatus	*none*	clause 22

16.55.6 Conversion between the parameters of the C function **BioAPI_GetIndicatorStatus** and the ASN.1 type **GetIndicatorStatus-ResponseParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 116.

Table 116 – Mapping between the parameters of the function BioAPI_GetIndicatorStatus and the ASN.1 type GetIndicatorStatus-ResponseParams

Function parameter	Component of the ASN.1 type	References
IndicatorStatus	indicatorStatus	clause 20 in conjunction with 15.45

16.56 Function BioAPI_GetLastErrorInfo

16.56.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_GetLastErrorInfo
        (BioAPI_ERROR_INFO *ErrorInfo);
```

16.56.2 There are no related BIP message types.

16.56.3 When a framework receives a call to the function **BioAPI_GetLastErrorInfo** from the local application, it shall set the members of the output parameter **ErrorInfo** (see 15.29) with information about the most recent error condition that caused a BioAPI function to return an error code. If the local application calls BioAPI functions on multiple threads, the framework shall return information about the last error occurred in a BioAPI function called on the same application thread on which this function has been called.

16.56.4 This function is mainly intended to provide diagnostic information to the local application. Since the possible values of the members of the **ErrorInfo** are not standardized and may vary across different platforms and implementations, normal BioAPI applications should not rely on those values for any important processing decisions.

16.57 Function BioAPI_Cancel

16.57.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Cancel
        (BioAPI_HANDLE BSPHandle);
```

16.57.2 A pair of BIP message types are related to this function: the **cancel** request BIP message type and the **cancel** response BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
Cancel-RequestParams ::= SEQUENCE {
        originalBSPHandle       BioAPI-HANDLE
}
```

and:

```
Cancel-ResponseParams ::= NULL
```

16.57.3 When a framework receives a call to the function **BioAPI_Cancel** from the local application, it shall first determine the hosting endpoint and the original BSP handle (say, *originalBSPHandle*) from the parameter **BSPHandle** as specified in clause 24. If the hosting endpoint is the local endpoint, then the framework shall make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, except that the parameter **BSPHandle** shall be set by converting from *originalBSPHandle* as specified in 15.42, and shall return to the local application the return value of the internal call. If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **cancel** request/response BIP message pair as specified in clause 27, using 16.57.5 to convert between function parameters and ASN.1 components when required within that clause. If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_BSP** to the local application.

16.57.4 When a framework receives (see 13.9) a **cancel** request BIP message from a master endpoint, it shall process the request via an internal BioAPI function call to **BioAPI_Cancel** to create and send a corresponding **cancel** response BIP message as specified in clause 28, using 16.57.5 to convert between function parameters and ASN.1 components when required within that clause.

16.57.5 Conversion between the parameters of the C function **BioAPI_Cancel** and the ASN.1 type **Cancel-RequestParams** shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 117.

Table 117 – Mapping between the parameters of the function BioAPI_Cancel and the ASN.1 type Cancel-RequestParams

Function parameter	Component of the ASN.1 type	References
BSPHandle	originalBSPHandle	clause 26

16.58 Function BioAPI_Free

16.58.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI BioAPI_Free
 (void *Ptr);
```

16.58.2 There are no related BIP message types.

16.58.3 When a framework receives a call to the function **BioAPI_Free** from the local application, it shall perform the following actions (in order):

a) search the **ApplicationOwnedMemoryBlocks** table (see 18.13) for an entry where the component **address** has the same value as the parameter **Ptr**;

b) if no such entry is found, then make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call, and return to the local application the return value of the internal call, skipping the following actions;

c) free the corresponding memory block;

d) delete the entry of the **ApplicationOwnedMemoryBlocks** table;

e) return the value 0 to the local application.

16.59 Function BioAPI_RegisterBSP

16.59.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI_RegisterBSP
        (const uint8_t *HostingEndpointIRI,
        const BioAPI_BSP_SCHEMA *BSPSchema,
        BioAPI_BOOL Update);
```

16.59.2 Three BIP message types are related to this function: the **registerBSP** request BIP message type, the **registerBSP** response BIP message type, and the **bspRegistrationEvent** notification BIP message type. These three BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
RegisterBSP-RequestParams ::= SEQUENCE {
                bspSchema           BioAPI-BSP-SCHEMA,
                update              BOOLEAN
        }

RegisterBSP-ResponseParams ::= NULL
```

and:

```
BSPRegistrationEvent-NotificationParams ::= SEQUENCE {
                bspSchema           BioAPI-BSP-SCHEMA,
                update              BOOLEAN
        }
```

16.59.3 When a framework receives a call to the function **BioAPI_RegisterBSP** from the local application, it shall first determine the hosting endpoint IRI by converting from the parameter **HostingEndpointIRI** as specified in 15.3, and then one of the three following subclauses applies.

16.59.3.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create a temporary abstract value (say, *incomingRequestParams*) of type **RegisterBSP-RequestParams** (see 16.59.2) by converting from the parameters of the **BioAPI_RegisterBSP** function call as specified in 16.59.6;

d) search the **VisibleBSPRegistrations** table (see 18.3) for an entry where:

1) the component **hostingEndpointIRI** contains the local endpoint IRI; and

2) the component **bspProductUuid** has the same value as the component **bspProductUuid** of the component **bspSchema** of *incomingRequestParams*;

e) if such an entry is found and the component **update** of *incomingRequestParams* has the value **FALSE**, then return the value **BioAPIERR_COMPONENT_ALREADY_REGISTERED** to the local application, skipping the remaining actions;

f) if such an entry is found and the component **update** of *incomingRequestParams* has the value **TRUE**, then delete the entry of the **VisibleBSPRegistrations** table (subclause 18.3.3 applies);

g) add an entry to the **VisibleBSPRegistrations** table (see 18.3), where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

2) the component **bspAccessUuid** shall be set to a dynamically generated UUID; and

3) the remaining components shall be set from the components of the component **bspSchema** of *incomingRequestParams* with the same names;

h) create a temporary abstract value (say, *outgoingNotificationParams*) of type **BSPRegistrationEvent-NotificationParams** (see 16.59.2), where all the components shall be set from the components of *incomingRequestParams* with the same names;

i) for each entry (say, *masterEndpoint*) of the **MasterEndpoints** table (see 18.1), create and send a **bspRegistrationEvent** notification BIP message (see 13.4) with the master endpoint IRI set from the component **masterEndpointIRI** of *masterEndpoint* and the parameter value set to *outgoingNotificationParams*;

j) return the value 0 to the local application.

16.59.3.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **registerBSP** request/response BIP message pair as specified in clause 27, using 16.59.6 to convert between function parameters and ASN.1 components when required within that clause.

16.59.3.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_ENDPOINT** to the local application.

16.59.4 When a framework receives (see 13.9) a **registerBSP** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **RegisterBSP-RequestParams** – see 16.59.2) of the **registerBSP** request BIP message;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_RegisterBSP**, where the parameters of the function call shall be set by converting from *incomingRequestParams* as specified in 16.59.6;

c) if the return value of the internal call is not 0, then create and send a corresponding **registerBSP** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

d) search the **VisibleBSPRegistrations** table (see 18.3) for an entry where:

1) the component **hostingEndpointIRI** contains the local endpoint IRI; and

2) the component **bspProductUuid** has the same value as the component **bspProductUuid** of the component **bspSchema** of *incomingRequestParams*;

e) if such an entry is found and the component **update** of *incomingRequestParams* has the value **FALSE**, then create and send a corresponding **registerBSP** response BIP message (see 13.3) with the return value set to **BioAPIERR_COMPONENT_ALREADY_REGISTERED**, skipping the remaining actions;

f) if such an entry is found and the component **update** of *incomingRequestParams* has the value **TRUE**, then delete the entry of the **VisibleBSPRegistrations** table (subclause 18.3.3 applies);

g) add an entry to the **VisibleBSPRegistrations** table (see 18.3), where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

2) the component **bspAccessUuid** shall be set to a dynamically generated UUID; and

3) the remaining components shall be set from the components of the component **bspSchema** of *incomingRequestParams* with the same names;

h) for each entry (say, *masterEndpoint*) of the **MasterEndpoints** table (see 18.1), create and send a **bspRegistrationEvent** notification BIP message (see 13.4) with the master endpoint IRI set from the component **masterEndpointIRI** of *masterEndpoint* and the parameter value set to *incomingRequestParams*;

i) create and send a corresponding **registerBSP** response BIP message (see 13.3) with the parameter value set to **NULL** and the return value set to 0.

16.59.5 When a framework receives (see 13.9) a **bspRegistrationEvent** notification BIP message from a slave endpoint, it shall perform the following actions (in order):

a) let *incomingNotificationParams* be the parameter value (of type **BSPRegistrationEvent-NotificationParams** – see 16.59.2) of the **bspRegistrationEvent** notification BIP message;

b) search the **VisibleBSPRegistrations** table (see 18.3) for an entry where:

1) the component **hostingEndpointIRI** contains the slave endpoint IRI; and

2) the component **bspProductUuid** has the same value as the component **bspProductUuid** of the component **bspSchema** of *incomingNotificationParams*;

c) if such an entry is found and the component **update** of *incomingNotificationParams* has the value **FALSE**, then skip the remaining actions;

d) if such an entry is found and the component **update** of *incomingNotificationParams* has the value **TRUE**, then delete the entry of the **VisibleBSPRegistrations** table (subclause 18.3.3 applies);

e) add an entry to the **VisibleBSPRegistrations** table (see 18.3), where:

1) the component **bspAccessUuid** shall be set to a dynamically generated UUID; and

2) the remaining components shall be set from the components of the component **bspSchema** of *incomingNotificationParams* with the same names;

f) conclude that no acknowledgement BIP message is to be sent.

16.59.6 Conversion between the parameters of the C function **BioAPI_RegisterBSP** and the ASN.1 type **RegisterBSP-RequestParams** (see 16.59.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 118.

Table 118 – Mapping between the parameters of the function BioAPI_RegisterBSP and the ASN.1 type RegisterBSP-RequestParams

Function parameter	Component of the ASN.1 type	References
HostingEndpointIRI	*none*	16.59.7
BSPSchema	bspSchema	clause 19 in conjunction with 15.19
Update	update	15.18

16.59.7 When converting from the parameters of the C function to the ASN.1 type, the parameter **HostingEndpointIRI** shall be ignored (it has already been used to determine the hosting endpoint). When converting from the ASN.1 type to the parameters of the C function, the parameter **HostingEndpointIRI** shall be set to **NULL** (indicating the local endpoint).

16.60 Function BioAPI_UnregisterBSP

16.60.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI_UnregisterBSP
        (const uint8_t *HostingEndpointIRI,
        const BioAPI_UUID *BSPProductUuid);
```

16.60.2 Three BIP message types are related to this function: the **unregisterBSP** request BIP message type, the **unregisterBSP** response BIP message type, and the **bspUnregistrationEvent** notification BIP message type. These three BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
UnregisterBSP-RequestParams ::= SEQUENCE {
        bspProductUuid          BioAPI-UUID
    }

UnregisterBSP-ResponseParams ::= NULL
```

and:

```
BSPUnregistrationEvent-NotificationParams ::= SEQUENCE {
        bspProductUuid          BioAPI-UUID
    }
```

16.60.3 When a framework receives a call to the function **BioAPI_UnregisterBSP** from the local application, it shall first determine the hosting endpoint by converting from the parameter **HostingEndpointIRI** as specified in 15.3, and then one of the three following subclauses applies.

16.60.3.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create a temporary abstract value (say, *incomingRequestParams*) of type **UnregisterBSP-RequestParams** (see 16.60.2) by converting from the parameters of the **BioAPI_UnregisterBSP** function call as specified in 16.60.6;

d) search the **VisibleBSPRegistrations** table (see 18.3) for an entry where:

1) the component **hostingEndpointIRI** contains the local endpoint IRI; and

2) the component **bspProductUuid** has the same value as the component **bspProductUuid** of *incomingRequestParams*;

e) if such an entry is not found, then return the value **BioAPIERR_NO_SUCH_COMPONENT_FOUND** to the local application, skipping the remaining actions;

f) delete the entry of the **VisibleBSPRegistrations** table (subclause 18.3.3 applies);

g) create a temporary abstract value (say, *outgoingNotificationParams*) of type **BSPUnregistrationEvent-NotificationParams** (see 16.60.2), where all the components shall be set from the components of *incomingRequestParams* with the same names;

h) for each entry (say, *masterEndpoint*) of the **MasterEndpoints** table (see 18.1), create and send a **bspUnregistrationEvent** notification BIP message (see 13.4) with the master endpoint IRI set from the component **masterEndpointIRI** of *masterEndpoint* and the parameter value set to *outgoingNotificationParams*;

i) return the value 0 to the local application.

16.60.3.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint an **unregisterBSP** request/response BIP message pair as specified in clause 27, using 16.60.6 to convert between function parameters and ASN.1 components when required within that clause.

16.60.3.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_ENDPOINT** to the local application.

16.60.4 When a framework receives (see 13.9) an **unregisterBSP** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **UnregisterBSP-RequestParams** – see 16.60.2) of the **unregisterBSP** request BIP message;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnregisterBSP**, where the parameters of the function call shall be set by converting from *incomingRequestParams* as specified in 16.60.6;

c) if the return value of the internal call is not 0, then create and send a corresponding **unregisterBSP** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

d) search the **VisibleBSPRegistrations** table (see 18.3) for an entry where:

1) the component **hostingEndpointIRI** contains the local endpoint IRI; and

2) the component **bspProductUuid** has the same value as the component **bspProductUuid** of *incomingRequestParams*;

e) if such an entry is not found, then create and send a corresponding **unregisterBSP** response BIP message (see 13.3) with the return value set to **BioAPIERR_NO_SUCH_COMPONENT_FOUND**, skipping the remaining actions;

f) delete the entry of the **VisibleBSPRegistrations** table (subclause 18.3.3 applies);

g) for each entry (say, *masterEndpoint*) of the **MasterEndpoints** table (see 18.1), create and send a **bspUnregistrationEvent** notification BIP message (see 13.4) with the master endpoint IRI set from the component **masterEndpointIRI** of *masterEndpoint* and the parameter value set to *incomingRequestParams*;

h) create and send a corresponding **unregisterBSP** response BIP message (see 13.3) with the parameter value set to **NULL** and the return value be set to 0.

16.60.5 When a framework receives (see 13.9) a **bspUnregistrationEvent** notification BIP message from a slave endpoint, it shall perform the following actions (in order):

a) let *incomingNotificationParams* be the parameter value (of type **BSPUnregistrationEvent-NotificationParams** – see 16.60.2) of the **bspUnregistrationEvent** notification BIP message;

b) search the **VisibleBSPRegistrations** table (see 18.3) for an entry where:

1) the component **hostingEndpointIRI** contains the slave endpoint IRI; and

2) the component **bspProductUuid** has the same value as the component **bspProductUuid** of *incomingNotificationParams*;

c) if such an entry is found, then delete the entry of the **VisibleBSPRegistrations** table (subclause 18.3.3 applies);

d) conclude that no acknowledgement BIP message is to be sent.

16.60.6 Conversion between the parameters of the C function **BioAPI_UnregisterBSP** and the ASN.1 type **UnregisterBSP-RequestParams** (see 16.60.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 119.

Table 119 – Mapping between the parameters of the function BioAPI_UnregisterBSP and the ASN.1 type UnregisterBSP-RequestParams

Function parameter	Component of the ASN.1 type	References
HostingEndpointIRI	*none*	16.60.7
BSPProductUuid	bspProductUuid	15.58

16.60.7 When converting from the parameters of the C function to the ASN.1 type, the parameter **HostingEndpointIRI** shall be ignored (it has already been used to determine the hosting endpoint). When converting from the ASN.1 type to the parameters of the C function, the parameter **HostingEndpointIRI** shall be set to **NULL** (indicating the local endpoint).

16.61 Function BioAPI_RegisterBFP

16.61.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI_RegisterBFP
        (const uint8_t *HostingEndpointIRI,
        const BioAPI_BFP_SCHEMA *BFPSchema,
        BioAPI_BOOL Update);
```

16.61.2 Three BIP message types are related to this function: the **registerBFP** request BIP message type, the **registerBFP** response BIP message type, and the **bfpRegistrationEvent** notification BIP message type. These three BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
RegisterBFP-RequestParams ::= SEQUENCE {
        bfpSchema               BioAPI-BFP-SCHEMA,
        update                  BOOLEAN
    }

RegisterBFP-ResponseParams ::= NULL
```

and:

```
BFPRegistrationEvent-NotificationParams ::= SEQUENCE {
        bfpSchema               BioAPI-BFP-SCHEMA,
        update                  BOOLEAN
    }
```

16.61.3 When a framework receives a call to the function **BioAPI_RegisterBFP** from the local application, it shall first determine the hosting endpoint by converting from the parameter **HostingEndpointIRI** as specified in 15.3, and then one of the three following subclauses applies.

16.61.3.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create a temporary abstract value (say, *incomingRequestParams*) of type **RegisterBFP-RequestParams** (see 16.61.2) by converting from the parameters of the **BioAPI_RegisterBFP** function call as specified in 16.61.6;

d) search the **VisibleBFPRegistrations** table (see 18.4) for an entry where:

1) the component **hostingEndpointIRI** contains the local endpoint IRI; and

2) the component **bfpProductUuid** has the same value as the component **bfpProductUuid** of the component **bfpSchema** of *incomingRequestParams*;

e) if such an entry is found and the component **update** of *incomingRequestParams* has the value **FALSE**, then return the value **BioAPIERR_COMPONENT_ALREADY_REGISTERED** to the local application, skipping the remaining actions;

f) if such an entry is found and the component **update** of *incomingRequestParams* has the value **TRUE**, then delete the entry of the **VisibleBFPRegistrations** table (subclause 18.4.3 applies);

g) add an entry to the **VisibleBFPRegistrations** table (see 18.4), where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI; and

2) the remaining components shall be set from the components of the component **bfpSchema** of *incomingRequestParams* with the same names;

h) create a temporary abstract value (say, *outgoingNotificationParams*) of type **BFPRegistrationEvent-NotificationParams** (see 16.61.2), where all the components shall be set from the components of *incomingRequestParams* with the same names;

i) for each entry (say, *masterEndpoint*) of the **MasterEndpoints** table (see 18.1), create and send a **bfpRegistrationEvent** notification BIP message (see 13.4) with the master endpoint IRI set from the component **masterEndpointIRI** of *masterEndpoint* and the parameter value set to *outgoingNotificationParams*;

j) return the value 0 to the local application.

16.61.3.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint a **registerBFP** request/response BIP message pair as specified in clause 27, using 16.61.6 to convert between function parameters and ASN.1 components when required within that clause.

16.61.3.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_ENDPOINT** to the local application.

16.61.4 When a framework receives (see 13.9) a **registerBFP** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **RegisterBFP-RequestParams** – see 16.61.2) of the **registerBFP** request BIP message;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_RegisterBFP**, where the parameters of the function call shall be set by converting from *incomingRequestParams* as specified in 16.61.6;

c) if the return value of the internal call is not 0, then create and send a corresponding **registerBFP** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

d) search the **VisibleBFPRegistrations** table (see 18.4) for an entry where:

1) the component **hostingEndpointIRI** contains the local endpoint IRI; and

2) the component **bfpProductUuid** has the same value as the component **bfpProductUuid** of the component **bfpSchema** of *incomingRequestParams*;

e) if such an entry is found and the component **update** of *incomingRequestParams* has the value **FALSE**, then create and send a corresponding **registerBFP** response BIP message (see 13.3) with the return value set to **BioAPIERR_NO_SUCH_COMPONENT_FOUND**, skipping the remaining actions;

f) such an entry is found and the component **update** of *incomingRequestParams* has the value **TRUE**, then delete the entry of the **VisibleBFPRegistrations** table (subclause 18.4.3 applies);

g) add an entry to the **VisibleBFPRegistrations** table (see 18.4), where:

1) the component **hostingEndpointIRI** shall be set to the local endpoint IRI; and

2) the remaining components shall be set from the components of the component **bfpSchema** of *incomingRequestParams* with the same names;

h) for each entry (say, *masterEndpoint*) of the **MasterEndpoints** table (see 18.1), create and send a **bfpRegistrationEvent** notification BIP message (see 13.4) with the master endpoint IRI set from the component **masterEndpointIRI** of *masterEndpoint* and the parameter value set to *incomingRequestParams*;

i) create and send a corresponding **registerBFP** response BIP message (see 13.3) with the parameter value set to **NULL** and the return value set to 0.

16.61.5 When a framework receives (see 13.9) a **bfpRegistrationEvent** notification BIP message from a slave endpoint, it shall perform the following actions (in order):

a) let *incomingNotificationParams* be the parameter value (of type **BFPRegistrationEvent-NotificationParams** – see 16.61.2) of the **bfpRegistrationEvent** notification BIP message;

b) search the **VisibleBFPRegistrations** table (see 18.4) for an entry where:

1) the component **hostingEndpointIRI** contains the slave endpoint IRI; and

2) the component **bfpProductUuid** has the same value as the component **bfpProductUuid** of the component **bfpSchema** of *incomingNotificationParams*;

c) if such an entry is found and the component **update** of *incomingNotificationParams* has the value **FALSE**, then skip the remaining actions;

d) if such an entry is found and the component **update** of *incomingNotificationParams* has the value **TRUE**, then delete the entry of the **VisibleBFPRegistrations** table (subclause 18.4.3 applies);

e) add an entry to the **VisibleBFPRegistrations** table (see 18.4), where all the components shall be set from the components of the component **bfpSchema** of *incomingNotificationParams* with the same names;

f) conclude that no acknowledgement BIP message is to be sent.

16.61.6 Conversion between the parameters of the C function **BioAPI_RegisterBFP** and the ASN.1 type **RegisterBFP-RequestParams** (see 16.61.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 120.

Table 120 – Mapping between the parameters of the function BioAPI_RegisterBFP and the ASN.1 type RegisterBFP-RequestParams

Function parameter	Component of the ASN.1 type	References
HostingEndpointIRI	*none*	16.61.7
BFPSchema	bfpSchema	clause 19 in conjunction with 15.5
Update	update	15.18

16.61.7 When converting from the parameters of the C function to the ASN.1 type, the parameter **HostingEndpointIRI** shall be ignored (it has already been used to determine the hosting endpoint). When converting from the ASN.1 type to the parameters of the C function, the parameter **HostingEndpointIRI** shall be set to **NULL** (indicating the local endpoint).

16.62 Function BioAPI_UnregisterBFP

16.62.1 This function is declared in BioAPI as follows:

```
BioAPI_RETURN BioAPI_UnregisterBFP
        (const uint8_t *HostingEndpointIRI,
        const BioAPI_UUID *BFPProductUuid);
```

16.62.2 Three BIP message types are related to this function: the **unregisterBFP** request BIP message type, the **unregisterBFP** response BIP message type, and the **bfpUnregistrationEvent** notification BIP message type. These three BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
UnregisterBFP-RequestParams ::= SEQUENCE {
        bfpProductUuid          BioAPI-UUID
    }

UnregisterBFP-ResponseParams ::= NULL
```

and:

```
BFPUnregistrationEvent-NotificationParams ::= SEQUENCE {
        bfpProductUuid          BioAPI-UUID
    }
```

16.62.3 When a framework receives a call to the function **BioAPI_UnregisterBFP** from the local application, it shall first determine the hosting endpoint by converting from the parameter **HostingEndpointIRI** as specified in 15.3, and then one of the three following subclauses applies.

16.62.3.1 If the hosting endpoint is the local endpoint, then the framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the same function with the same parameter values as the incoming call;

b) if the return value of the internal call is not 0, then return that value to the local application, skipping the remaining actions;

c) create a temporary abstract value (say, *incomingRequestParams*) of type **UnregisterBFP-RequestParams** (see 16.62.2) by converting from the parameters of the **BioAPI_UnregisterBFP** function call as specified in 16.62.6;

d) search the **VisibleBFPRegistrations** table (see 18.4) for an entry where:

1) the component **hostingEndpointIRI** contains the local endpoint IRI; and

2) the component **bfpProductUuid** has the same value as the component **bfpProductUuid** of *incomingRequestParams*;

e) if such an entry is not found, then return the value **BioAPIERR_NO_SUCH_COMPONENT_FOUND** to the local application, skipping the remaining actions;

f) delete the entry of the **VisibleBFPRegistrations** table (subclause 18.4.3 applies);

g) create a temporary abstract value (say, *outgoingNotificationParams*) of type **BFPUnregistrationEvent-NotificationParams** (see 16.62.2), where all the components shall be set from the components of *incomingRequestParams* with the same names;

h) for each entry (say, *masterEndpoint*) of the **MasterEndpoints** table (see 18.1), create and send a **bfpUnregistrationEvent** notification BIP message (see 13.4) with the master endpoint IRI set from the component **masterEndpointIRI** of *masterEndpoint* and the parameter value set to *outgoingNotificationParams*;

i) return the value 0 to the local application.

16.62.3.2 If the hosting endpoint is a slave endpoint of the framework, then the framework shall process the call by exchanging with the hosting endpoint an **unregisterBFP** request/response BIP message pair as specified in clause 27, using 16.62.6 to convert between function parameters and ASN.1 components when required within that clause.

16.62.3.3 If the hosting endpoint cannot be determined, then the framework shall return the value **BioAPIERR_UNABLE_TO_LOCATE_ENDPOINT** to the local application.

16.62.4 When a framework receives (see 13.9) an **unregisterBFP** request BIP message from a master endpoint, it shall perform the following actions (in order):

a) let *incomingRequestParams* be the parameter value (of type **UnregisterBFP-RequestParams** – see 16.62.2) of the **unregisterBFP** request BIP message;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnregisterBFP**, where the parameters of the function call shall be set by converting from *incomingRequestParams* as specified in 16.62.6;

c) if the return value of the internal call is not 0, then create and send a corresponding **unregisterBFP** response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

d) search the **VisibleBFPRegistrations** table (see 18.4) for an entry where:

1) the component **hostingEndpointIRI** contains the local endpoint IRI; and

2) the component **bfpProductUuid** has the same value as the component **bfpProductUuid** of *incomingRequestParams*;

e) if such an entry is not found, then create and send a corresponding **unregisterBFP** response BIP message (see 13.3) with the return value set to **BioAPIERR_NO_SUCH_COMPONENT_FOUND**, skipping the remaining actions;

f) delete the entry of the **VisibleBFPRegistrations** table (subclause 18.4.3 applies);

g) for each entry (say, *masterEndpoint*) of the **MasterEndpoints** table (see 18.1), create and send a **bfpUnregistrationEvent** notification BIP message (see 13.4) with the master endpoint IRI set from the component **masterEndpointIRI** of *masterEndpoint* and the parameter value set to *incomingRequestParams*;

h) create and send a corresponding **unregisterBFP** response BIP message (see 13.3) with the parameter value set to **NULL** and the return value set to 0.

16.62.5 When a framework receives (see 13.9) a **bfpUnregistrationEvent** notification BIP message from a slave endpoint, it shall perform the following actions (in order):

a) let *incomingNotificationParams* be the parameter value (of type **BFPUnregistrationEvent-NotificationParams** – see 16.62.2) of the **bfpUnregistrationEvent** notification BIP message;

b) search the **VisibleBFPRegistrations** table (see 18.4) for an entry where:

1) the component **hostingEndpointIRI** contains the slave endpoint IRI; and

2) the component **bfpProductUuid** has the same value as the component **bfpProductUuid** of *incomingNotificationParams*;

c) if such an entry is found, then delete the entry of the **VisibleBFPRegistrations** table (see 18.4.3);

d) conclude that no acknowledgement BIP message is to be sent.

16.62.6 Conversion between the parameters of the C function **BioAPI_UnregisterBFP** and the ASN.1 type **UnregisterBFP-RequestParams** (see 16.62.2) shall be done by converting between individual function parameters and ASN.1 components in accordance with Table 121.

Table 121 – Mapping between the parameters of the function BioAPI_UnregisterBFP and the ASN.1 type UnregisterBFP-RequestParams

Function parameter	Component of the ASN.1 type	References
HostingEndpointIRI	*none*	16.62.7
BFPProductUuid	bfpProductUuid	15.58

16.62.7 When converting from the parameters of the C function to the ASN.1 type, the parameter **HostingEndpointIRI** shall be ignored (it has already been used to determine the hosting endpoint). When converting from the ASN.1 type to the parameters of the C function, the parameter **HostingEndpointIRI** shall be set to **NULL** (indicating the local endpoint).

17 Callback functions defined in BioAPI and corresponding BIP messages

17.1 Callback function BioAPI_EVENT_HANDLER

17.1.1 The C function pointer type for this callback function is defined in BioAPI as follows:

```
typedef BioAPI_RETURN (*BioAPI_EVENT_HANDLER)
        (const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_ID UnitID,
        void *EventHandlerCtx,
        const BioAPI_UNIT_SCHEMA *UnitSchema,
        BioAPI_EVENT EventType);
```

17.1.2 The corresponding C function pointer type in the BioSPI interface is defined in BioAPI as follows:

```
typedef BioAPI_RETURN (*BioSPI_EVENT_HANDLER)
        (const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_ID UnitID,
        const BioAPI_UNIT_SCHEMA *UnitSchema,
        BioAPI_EVENT EventType);
```

17.1.3 One BIP message type is related to this function: the **unitEvent** notification BIP message type. This BIP message type carries a value of the following BIP message parameter ASN.1 type:

```
UnitEvent-NotificationParams ::= SEQUENCE {
        bspProductUuid          BioAPI-UUID,
        unitID                  BioAPI-UNIT-ID,
        unitSchema              BioAPI-UNIT-SCHEMA OPTIONAL,
        unitEventType           BioAPI-UNIT-EVENT-TYPE
}
```

17.1.4 The following ASN.1 types are defined to aid the specification of the behaviour of a framework, but their abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
UnitEventHandlerCallbackParams ::= SEQUENCE {
        unitEventHandlerAddress     MemoryAddress,
        unitEventHandlerContext     MemoryAddress,
        bspUuid                     BioAPI-UUID,
        unitID                      BioAPI-UNIT-ID,
        unitSchema                  BioAPI-UNIT-SCHEMA OPTIONAL,
        unitEventType               BioAPI-UNIT-EVENT-TYPE
}
UnitEventInfo ::= SEQUENCE {
        hostingEndpointIRI          EndpointIRI,
        bspProductUuid              BioAPI-UUID,
        unitID                      BioAPI-UNIT-ID,
        unitSchema                  BioAPI-UNIT-SCHEMA OPTIONAL,
        unitEventType               BioAPI-UNIT-EVENT-TYPE
}
```

17.1.5 When a framework receives a call to the callback function **BioSPI_EVENT_HANDLER** from a BSP, it shall perform the following actions (in order):

a) create a temporary abstract value (say, *incomingNotificationParams*) of type **UnitEvent-NotificationParams** (see 17.1.3) by converting from the parameters of the function call as specified in 17.1.7;

b) create a temporary abstract value (say, *eventInfo*) of type **UnitEventInfo** (see 17.1.4), where the component **hostingEndpointIRI** shall be set to the local endpoint IRI, and the remaining components shall be set from the components of *incomingNotificationParams* with the same names;

c) notify the unit event based on *eventInfo* to zero or more subscribers (local application unit event handlers or master endpoints) as specified in clause 29; and

d) return the value 0 to the BSP.

17.1.6 When a framework receives (see 13.9) a **unitEvent** notification BIP message from a slave endpoint, it shall perform the following actions (in order):

a) let *incomingNotificationParams* be the parameter value (of type **UnitEvent-NotificationParams** – see 17.1.3) of the **unitEvent** notification BIP message;

b) create a temporary abstract value (say, *eventInfo*) of type **UnitEventInfo** (see 17.1.4), where the component **hostingEndpointIRI** shall be set from the component **slaveEndpointIRI** of the notification BIP message, and the remaining components shall be set from the components of *incomingNotificationParams* with the same names;

c) notify the unit event based on *eventInfo* to zero or more subscribers (local application unit event handlers or master endpoints) as specified in clause 29;

d) conclude that no acknowledgement BIP message is to be sent.

17.1.7 Conversion from the parameters of the C callback function **BioSPI_EVENT_HANDLER** to the ASN.1 type **UnitEvent-NotificationParams** (see 17.1.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 122.

Table 122 – Mapping from the parameters of the callback function BioSPI_EVENT_HANDLER to the ASN.1 type UnitEvent-NotificationParams

Function parameter	Component of the ASN.1 type	References
BSPUuid	bspProductUuid	clause 19 in conjunction with 15.58
UnitID	unitID	15.55
UnitSchema	unitSchema	clause 19 in conjunction with 15.57
EventType	unitEventType	15.30

17.1.8 Conversion from the ASN.1 type **UnitEventHandlerCallbackParams** (see 17.1.4) to the parameters of the C callback function **BioAPI_EVENT_HANDLER** shall be done by converting from individual ASN.1 components to function parameters in accordance with Table 123.

Table 123 – Mapping from the ASN.1 type UnitEventHandlerCallbackParams to the parameters of the callback function BioAPI_EVENT_HANDLER

Component of the ASN.1 type	Function parameter	References
unitEventHandlerAddress	*none*	17.1.9
unitEventHandlerContext	EventHandlerCtx	15.1.7
bspUuid	BSPUuid	clause 19 in conjunction with 15.58
unitID	UnitID	15.55
unitSchema	UnitSchema	clause 19 in conjunction with 15.57
unitEventType	EventType	15.30

17.1.9 The component **unitEventHandlerAddress** corresponds to the callback address (rather than to a parameter) of the callback function. Conversion shall be done by applying 15.1.7.

17.2 Callback function BioAPI_GUI_SELECT_EVENT_HANDLER

17.2.1 The C function pointer type for this callback function is defined in BioAPI as follows:

```
typedef BioAPI_RETURN (BioAPI *BioAPI_GUI_SELECT_EVENT_HANDLER)
        (const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_ID UnitID,
        const BioAPI_HANDLE *BSPHandle,
        BioAPI_GUI_ENROLL_TYPE EnrollType,
        void *GUISelectEventHandlerCtx,
        BioAPI_GUI_OPERATION Operation,
        BioAPI_GUI_MOMENT Moment,
        BioAPI_RETURN ResultCode,
        uint32_t MaxNumEnrollSamples,
        BioAPI_BIR_SUBTYPE_MASK SelectableInstances,
        BioAPI_BIR_SUBTYPE_MASK *SelectedInstances,
        BioAPI_BIR_SUBTYPE_MASK CapturedInstances,
        const uint8_t *Text,
        BioAPI_GUI_RESPONSE *Response);
```

17.2.2 The corresponding C function pointer type in the BioSPI interface is defined in BioAPI as follows:

```
typedef BioAPI_RETURN (BioAPI *BioSPI_GUI_SELECT_EVENT_HANDLER)
        (const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_ID UnitID,
        const BioAPI_HANDLE *BSPHandle,
        BioAPI_GUI_ENROLL_TYPE EnrollType,
        BioAPI_GUI_OPERATION Operation,
        BioAPI_GUI_MOMENT Moment,
        BioAPI_RETURN ResultCode,
        uint32_t MaxNumEnrollSamples,
        BioAPI_BIR_SUBTYPE_MASK SelectableInstances,
        BioAPI_BIR_SUBTYPE_MASK *SelectedInstances,
        BioAPI_BIR_SUBTYPE_MASK CapturedInstances,
        const uint8_t *Text,
        BioAPI_GUI_RESPONSE *Response);
```

17.2.3 A pair of BIP message types are related to this function: the **guiSelectEvent** notification BIP message type and the **guiSelectEvent** acknowledgement BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
GUISelectEvent-NotificationParams ::= SEQUENCE {
        guiEventSubscriptionUuid        BioAPI-UUID OPTIONAL,
        bspProductUuid                  BioAPI-UUID,
        unitID                          BioAPI-UNIT-ID,
        originalBSPHandle               BioAPI-HANDLE OPTIONAL,
        enrollType                      BioAPI-GUI-ENROLL-TYPE,
        operation                       BioAPI-GUI-OPERATION,
        moment                          BioAPI-GUI-MOMENT,
        resultCode                      BioAPI-RETURN,
        maxNumEnrollSamples             UnsignedInt,
        selectableInstances             BioAPI-BIR-SUBTYPE-MASK,
        capturedInstances               BioAPI-BIR-SUBTYPE-MASK,
        text                            UTF8String OPTIONAL
    }
```

and:

```
GUISelectEvent-AcknowledgementParams ::= SEQUENCE {
        selectedInstances               BioAPI-BIR-SUBTYPE-MASK,
        response                        BioAPI-GUI-RESPONSE
    }
```

17.2.4 The following ASN.1 types are defined to aid the specification of the behaviour of a framework, but their abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
GUISelectEventHandlerCallbackParams ::= SEQUENCE {
        guiSelectEventHandlerAddress    MemoryAddress,
        guiSelectEventHandlerContext    MemoryAddress,
        bspUuid                         BioAPI-UUID,
        unitID                          BioAPI-UNIT-ID,
        bspHandle                       BioAPI-HANDLE OPTIONAL,
        enrollType                      BioAPI-GUI-ENROLL-TYPE,
        operation                       BioAPI-GUI-OPERATION,
        moment                          BioAPI-GUI-MOMENT,
        resultCode                      BioAPI-RETURN,
        maxNumEnrollSamples             UnsignedInt,
        selectableInstances             BioAPI-BIR-SUBTYPE-MASK,
        capturedInstances               BioAPI-BIR-SUBTYPE-MASK,
        text                            UTF8String OPTIONAL
    }
GUISelectEventInfo ::= SEQUENCE {
        subscriberEndpointIRI           EndpointIRI,
        guiEventSubscriptionUuid        BioAPI-UUID OPTIONAL,
        hostingEndpointIRI              EndpointIRI,
        bspProductUuid                  BioAPI-UUID,
        unitID                          BioAPI-UNIT-ID,
        originalBSPHandle               BioAPI-HANDLE OPTIONAL,
        enrollType                      BioAPI-GUI-ENROLL-TYPE,
        operation                       BioAPI-GUI-OPERATION,
        moment                          BioAPI-GUI-MOMENT,
        resultCode                      BioAPI-RETURN,
        maxNumEnrollSamples             UnsignedInt,
        selectableInstances             BioAPI-BIR-SUBTYPE-MASK,
        capturedInstances               BioAPI-BIR-SUBTYPE-MASK,
        text                            UTF8String OPTIONAL
    }
```

17.2.5 When a framework receives a call to the callback function **BioSPI_GUI_SELECT_EVENT_HANDLER** from a BSP, it shall perform the following actions (in order):

a) create a temporary abstract value (say, *incomingNotificationParams*) of type **GUISelectEvent-NotificationParams** (see 17.2.3) by converting from the parameters of the function call as specified in 17.2.7;

b) search the **GUIEventRedirectors** table (see 18.12) for an entry where the component **originalBSPHandle** has the same value as the component **originalBSPHandle** of *incomingNotificationParams* and the component **GUISelectEventRedirected** has the value **TRUE**;

c) if there is no matching entry, then create a temporary abstract value (say, *eventInfo*) of type **GUISelectEventInfo** (see 17.2.4) where:

1) if the **AttachSessionRemoteReferences** table (see 18.9) has an entry where the component **originalBSPHandle** has the same value as the component **originalBSPHandle** of *incomingNotificationParams*, then the component **subscriberEndpointIRI** of *eventInfo* shall be set from the component **referrerEndpointIRI** of that entry; otherwise, it shall be set to the local endpoint IRI;

2) the optional component **guiEventSubscriptionUuid** shall be absent;

3) the component **hostingEndpointIRI** shall be set to the local endpoint IRI; and

4) the remaining components shall be set from the components of *incomingNotificationParams* with the same names;

d) if there is one matching entry (say, *redirector*), then create a temporary abstract value (say, *eventInfo*) of type **GUISelectEventInfo** (see 17.2.4) where:

1) the component **subscriberEndpointIRI** shall be set from the component **subscriberEndpointIRI** of *redirector*;

2) the optional component **guiEventSubscriptionUuid** shall be present and shall be set from the component **guiEventSubscriptionUuid** of *redirector*;

3) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

4) the optional component **originalBSPHandle** shall be absent; and

5) the remaining components shall be set from the components of *incomingNotificationParams* with the same names;

e) notify a GUI select event based on *eventInfo* to a subscriber (either a GUI select event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 30;

f) if *incomingReturnValue* is not 0, then return that value to the BSP, skipping the remaining actions;

g) set the output parameters of the call to the callback function **BioSPI_GUI_SELECT_EVENT_HANDLER** by converting from *incomingAcknowledgementParams* as specified in 17.2.9; and

h) return the value 0 to the BSP.

17.2.6 When a framework receives (see 13.9) a **guiSelectEvent** notification BIP message from a slave endpoint, it shall perform the following actions (in order):

a) let *incomingNotificationParams* be the parameter value (of type **GUISelectEvent-NotificationParams** – see 17.2.3) of the notification BIP message;

b) create a temporary abstract value (say, *eventInfo*) of type **GUISelectEventInfo** (see 17.2.4) where:

1) the components **subscriberEndpointIRI** shall be set to the local endpoint IRI;

2) the component **hostingEndpointIRI** shall be set from the component **slaveEndpointIRI** of the notification BIP message; and

3) the remaining components shall be set from the components of *incomingNotificationParams* with the same names;

c) notify a GUI select event based on *eventInfo* to a subscriber (either a GUI select event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 30;

d) if *incomingReturnValue* is not 0, then create and send a corresponding **guiSelectEvent** acknowledgement BIP message (see 13.5) with the return value set to that value, skipping the remaining actions;

e) create and send a corresponding **guiSelectEvent** acknowledgement BIP message (see 13.5) with the parameter value set to *incomingAcknowledgementParams* and the return value set to 0.

17.2.7 Conversion from the parameters of the C callback function **BioSPI_GUI_SELECT_EVENT_HANDLER** to the ASN.1 type **GUISelectEvent-NotificationParams** (see 17.2.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 124.

Table 124 – Mapping from the parameters of the callback function BioSPI_GUI_SELECT_EVENT_HANDLER to the ASN.1 type GUISelectEvent-NotificationParams

Function parameter	Component of the ASN.1 type	References
none	guiEventSubscriptionUuid	17.2.8
BSPUuid	bspProductUuid	clause 19 in conjunction with 15.58
UnitID	unitID	15.55
BSPHandle	originalBSPHandle	clause 19 in conjunction with 15.42
EnrollType	enrollType	15.38
Operation	operation	15.39
Moment	moment	15.37
ResultCode	resultCode	15.52
MaxNumEnrollSamples	maxNumEnrollSamples	15.1.6
SelectableInstances	selectableInstances	15.17
CapturedInstances	capturedInstances	15.17
Text	text	15.2
SelectedInstances	*none*	clause 22
Response	*none*	clause 22

17.2.8 The optional component **guiEventSubscriptionUuid** shall be absent.

17.2.9 Conversion from the ASN.1 type **GUISelectEvent-AcknowledgementParams** (see 17.2.3) to the parameters of the C callback function **BioSPI_GUI_SELECT_EVENT_HANDLER** shall be done by converting from individual ASN.1 components to function parameters in accordance with Table 125.

Table 125 – Mapping from the ASN.1 type GUISelectEvent-AcknowledgementParams to the parameters of the callback function BioSPI_GUI_SELECT_EVENT_HANDLER

Component of the ASN.1 type	Function parameter	References
selectedInstances	SelectedInstances	clause 20 in conjunction with 15.17
response	Response	clause 20 in conjunction with 15.40

17.2.10 Conversion from the ASN.1 type **GUISelectEventHandlerCallbackParams** (see 17.2.4) to the parameters of the C callback function **BioAPI_GUI_SELECT_EVENT_HANDLER** shall be done by converting from individual ASN.1 components to function parameters with Table 126.

Table 126 – Mapping from the ASN.1 type GUISelectEventHandlerCallbackParams to the parameters of the callback function BioAPI_GUI_SELECT_EVENT_HANDLER

Component of the ASN.1 type	Function parameter	References
guiSelectEventHandlerAddress	*none*	17.2.11
guiSelectEventHandlerContext	GUISelectEventHandlerCtx	15.1.7
bspUuid	BSPUuid	clause 19 in conjunction with 15.58
unitID	UnitID	15.55
bspHandle	BSPHandle	clause 19 in conjunction with 15.42
operation	Operation	15.39
moment	Moment	15.37
resultCode	ResultCode	15.52
maxNumEnrollSamples	MaxNumEnrollSamples	15.1.6
selectableInstances	SelectableInstances	15.17
capturedInstances	CapturedInstances	15.17
text	Text	15.2
none	SelectedInstances	clause 22
none	Response	clause 22

17.2.11 The component **guiSelectEventHandlerAddress** corresponds to the address of the function rather than to a parameter of the function. Conversion shall be done by applying 15.1.7.

17.2.12 Conversion from the parameters of the C callback function **BioAPI_GUI_SELECT_EVENT_HANDLER** to the ASN.1 type **GUISelectEvent-AcknowledgementParams** (see 17.2.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 127.

Table 127 – Mapping from the parameters of the callback function BioAPI_GUI_SELECT_EVENT_HANDLER to the ASN.1 type GUISelectEvent-AcknowledgementParams

Function parameter	Component of the ASN.1 type	References
SelectedInstances	selectedInstances	clause 20 in conjunction with 15.17
Response	response	clause 20 in conjunction with 15.40

17.3 Callback function BioAPI_GUI_STATE_EVENT_HANDLER

17.3.1 The C function pointer type for this callback function is defined in BioAPI as follows:

```
typedef BioAPI_RETURN (BioAPI *BioAPI_GUI_STATE_EVENT_HANDLER)
        (const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_ID UnitID,
        const BioAPI_HANDLE *BSPHandle,
        void *GUIStateEventHandlerCtx,
        BioAPI_GUI_OPERATION Operation,
        BioAPI_GUI_SUBOPERATION Suboperation,
        BioAPI_BIR_PURPOSE Purpose,
        BioAPI_GUI_MOMENT Moment,
        BioAPI_RETURN ResultCode,
        int32_t EnrollSampleIndex,
        const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
        const uint8_t *Text,
        BioAPI_GUI_RESPONSE *Response,
        int32_t *EnrollSampleIndexToRecapture);
```

17.3.2 The corresponding C function pointer type in the BioSPI interface is defined in BioAPI as follows:

```
typedef BioAPI_RETURN (BioAPI *BioSPI_GUI_STATE_EVENT_HANDLER)
        (const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_ID UnitID,
        const BioAPI_HANDLE *BSPHandle,
        BioAPI_GUI_OPERATION Operation,
        BioAPI_GUI_SUBOPERATION Suboperation,
        BioAPI_BIR_PURPOSE Purpose,
        BioAPI_GUI_MOMENT Moment,
        BioAPI_RETURN ResultCode,
        int32_t EnrollSampleIndex,
        const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
        const uint8_t *Text,
        BioAPI_GUI_RESPONSE *Response,
        int32_t *EnrollSampleIndexToRecapture);
```

17.3.3 A pair of BIP message types are related to this function: the **guiStateEvent** notification BIP message type and the **guiStateEvent** acknowledgement BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
GUIStateEvent-NotificationParams ::= SEQUENCE {
        guiEventSubscriptionUuid    BioAPI-UUID OPTIONAL,
        bspProductUuid              BioAPI-UUID,
        unitID                      BioAPI-UNIT-ID,
        originalBSPHandle           BioAPI-HANDLE OPTIONAL,
        operation                   BioAPI-GUI-OPERATION,
        suboperation                BioAPI-GUI-SUBOPERATION,
        purpose                     BioAPI-BIR-PURPOSE,
```

```
        moment                          BioAPI-GUI-MOMENT,
        resultCode                      BioAPI-RETURN,
        enrollSampleIndex               SignedInt,
        bitmaps                         BioAPI-GUI-BITMAP-ARRAY OPTIONAL,
        text                            UTF8String OPTIONAL
    }
```

and:

```
GUIStateEvent-AcknowledgementParams ::= SEQUENCE {
        response                        BioAPI-GUI-RESPONSE,
        enrollSampleIndexToRecapture    SignedInt
    }
```

17.3.4 The following ASN.1 types are defined to aid the specification of the behaviour of a framework, but their abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
GUIStateEventHandlerCallbackParams ::= SEQUENCE {
        guiStateEventHandlerAddress     MemoryAddress,
        guiStateEventHandlerContext     MemoryAddress,
        bspUuid                         BioAPI-UUID,
        unitID                          BioAPI-UNIT-ID,
        bspHandle                       BioAPI-HANDLE OPTIONAL,
        operation                       BioAPI-GUI-OPERATION,
        suboperation                    BioAPI-GUI-SUBOPERATION,
        purpose                         BioAPI-BIR-PURPOSE,
        moment                          BioAPI-GUI-MOMENT,
        resultCode                      BioAPI-RETURN,
        enrollSampleIndex               SignedInt,
        bitmaps                         BioAPI-GUI-BITMAP-ARRAY OPTIONAL,
        text                            UTF8String OPTIONAL
    }
GUIStateEventInfo ::= SEQUENCE {
        subscriberEndpointIRI           EndpointIRI,
        guiEventSubscriptionUuid        BioAPI-UUID OPTIONAL,
        hostingEndpointIRI              EndpointIRI,
        bspProductUuid                  BioAPI-UUID,
        unitID                          BioAPI-UNIT-ID,
        originalBSPHandle               BioAPI-HANDLE OPTIONAL,
        operation                       BioAPI-GUI-OPERATION,
        suboperation                    BioAPI-GUI-SUBOPERATION,
        purpose                         BioAPI-BIR-PURPOSE,
        moment                          BioAPI-GUI-MOMENT,
        resultCode                      BioAPI-RETURN,
        enrollSampleIndex               SignedInt,
        bitmaps                         BioAPI-GUI-BITMAP-ARRAY OPTIONAL,
        text                            UTF8String OPTIONAL
    }
```

17.3.5 When a framework receives a call to the callback function **BioSPI_GUI_STATE_EVENT_HANDLER** from a BSP, it shall perform the following actions (in order):

a) create a temporary abstract value (say, *incomingNotificationParams*) of type **GUIStateEvent-NotificationParams** (see 17.3.3) by converting from the parameters of the function call as specified in 17.3.7;

b) search the **GUIEventRedirectors** table (see 18.12) for an entry where the component **originalBSPHandle** has the same value as the component **originalBSPHandle** of *incomingNotificationParams* and the component **guiStateEventRedirected** has the value **TRUE**;

c) if there is no matching entry, then create a temporary abstract value (say, *eventInfo*) of type **GUIStateEventInfo** (see 17.3.4) where:

   1) if the **AttachSessionRemoteReferences** table (see 18.9) has an entry where the component **originalBSPHandle** has the same value as the component **originalBSPHandle** of *incomingNotificationParams*, then the component **subscriberEndpointIRI** of *eventInfo* shall be set from the component **referrerEndpointIRI** of that entry; otherwise, it shall be set to the local endpoint IRI;

   2) the optional component **guiEventSubscriptionUuid** shall be absent;

3) the component **hostingEndpointIRI** shall be set to the local endpoint IRI; and

4) the remaining components shall be set from the components of *incomingNotificationParams* with the same names;

d) if there is one matching entry (say, *redirector*), then create a temporary abstract value (say, *eventInfo*) of type **GUIStateEventInfo** (see 17.3.4) where:

1) the component **subscriberEndpointIRI** shall be set from the component **subscriberEndpointIRI** of *redirector*;

2) the optional component **guiEventSubscriptionUuid** shall be present and shall be set from the component **guiEventSubscriptionUuid** of *redirector*;

3) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

4) the optional component **originalBSPHandle** shall be absent; and

5) the remaining components shall be set from the components of *incomingNotificationParams* with the same names;

e) notify a GUI state event based on *eventInfo* to a subscriber (either a GUI state event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 31;

f) if *incomingReturnValue* is not 0, then return that value to the BSP, skipping the remaining actions;

g) set the output parameters of the call to the callback function **BioSPI_GUI_STATE_EVENT_HANDLER** by converting from *incomingAcknowledgementParams* as specified in 17.3.9; and

h) return the value 0 to the BSP.

17.3.6 When a framework receives (see 13.9) a **guiStateEvent** notification BIP message from a slave endpoint, it shall perform the following actions (in order):

a) let *incomingNotificationParams* be the parameter value (of type **GUIStateEvent-NotificationParams** – see 17.3.3) of the notification BIP message;

b) create a temporary abstract value (say, *eventInfo*) of type **GUIStateEventInfo** (see 17.3.4) where:

1) the components **subscriberEndpointIRI** shall be set to the local endpoint IRI;

2) the component **hostingEndpointIRI** shall be set from the component **slaveEndpointIRI** of the notification BIP message; and

3) the remaining components shall be set from the components of *incomingNotificationParams* with the same names;

c) notify a GUI state event based on *eventInfo* to a subscriber (either a GUI state event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 31;

d) if *incomingReturnValue* is not 0, then create and send a corresponding **guiStateEvent** acknowledgement BIP message (see 13.5) with the return value set to that value, skipping the remaining actions;

e) create and send a corresponding **guiStateEvent** acknowledgement BIP message (see 13.5) with the parameter value set to *incomingAcknowledgementParams* and the return value set to 0.

17.3.7 Conversion from the parameters of the C callback function **BioSPI_GUI_STATE_EVENT_HANDLER** to the ASN.1 type **GUIStateEvent-NotificationParams** (see 17.3.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 128.

Table 128 – Mapping from the parameters of the callback function BioSPI_GUI_STATE_EVENT_HANDLER to the ASN.1 type GUIStateEvent-NotificationParams

Function parameter	Component of the ASN.1 type	References
none	guiEventSubscriptionUuid	17.3.8
BSPUuid	bspProductUuid	clause 19 in conjunction with 15.58
UnitID	unitID	15.55
BSPHandle	originalBSPHandle	clause 19 in conjunction with 15.42
Operation	operation	15.39
Suboperation	suboperation	15.41
Purpose	purpose	15.14
Moment	moment	15.37
ResultCode	resultCode	15.52
EnrollSampleIndex	enrollSampleIndex	15.1.6
Bitmaps	bitmaps	clause 19 in conjunction with 15.35
Text	text	15.2
Response	*none*	clause 22
EnrollSampleIndexToRecapture	*none*	clause 22

17.3.8 The optional component **guiEventSubscriptionUuid** shall be absent.

17.3.9 Conversion from the ASN.1 type **GUIStateEvent-AcknowledgementParams** (see 17.3.3) to the parameters of the C callback function **BioSPI_GUI_STATE_EVENT_HANDLER** shall be done by converting from individual ASN.1 components to function parameters in accordance with Table 129.

Table 129 – Mapping from the ASN.1 type GUIStateEvent-AcknowledgementParams to the parameters of the callback function BioSPI_GUI_STATE_EVENT_HANDLER

Component of the ASN.1 type	Function parameter	References
response	Response	clause 20 in conjunction with 15.40
enrollSampleIndexToRecapture	EnrollSampleIndexToRecapture	clause 20 in conjunction with 15.1.6

17.3.10 Conversion from the ASN.1 type **GUIStateEventHandlerCallbackParams** (see 17.3.4) to the parameters of the C callback function **BioAPI_GUI_STATE_EVENT_HANDLER** shall be done by converting from individual ASN.1 components to function parameters with Table 130.

Table 130 – Mapping from the ASN.1 type GUIStateEventHandlerCallbackParams to the parameters of the callback function BioAPI_GUI_STATE_EVENT_HANDLER

Component of the ASN.1 type	Function parameter	References
guiStateEventHandlerAddress	*none*	17.3.11
guiStateEventHandlerContext	GUIStateEventHandlerCtx	15.1.7
bspUuid	BSPUuid	clause 19 in conjunction with 15.58
unitID	UnitID	15.55
bspHandle	BSPHandle	clause 19 in conjunction with 15.42
operation	Operation	15.39
suboperation	Suboperation	15.41
purpose	Purpose	15.14
moment	Moment	15.37
resultCode	ResultCode	15.52
enrollSampleIndex	EnrollSampleIndex	15.1.6
bitmaps	Bitmaps	clause 19 in conjunction with 15.35
text	Text	15.2
none	Response	clause 22
none	EnrollSampleIndexToRecapture	clause 22

17.3.11 The component **guiStateEventHandlerAddress** corresponds to the address of the function rather than to a parameter of the function. Conversion shall be done by applying 15.1.7.

17.3.12 Conversion from the parameters of the C callback function **BioAPI_GUI_STATE_EVENT_HANDLER** to the ASN.1 type **GUIStateEvent-AcknowledgementParams** (see 17.3.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 131.

Table 131 – Mapping from the parameters of the callback function BioAPI_GUI_STATE_EVENT_HANDLER to the ASN.1 type GUIStateEvent-AcknowledgementParams

Function parameter	Component of the ASN.1 type	References
Response	response	clause 20 in conjunction with 15.40
EnrollSampleIndexToRecapture	enrollSampleIndexToRecapture	clause 20 in conjunction with 15.1.6

17.4 Callback function BioAPI_GUI_PROGRESS_EVENT_HANDLER

17.4.1 The C function pointer type for this callback function is defined in BioAPI as follows:

```
typedef BioAPI_RETURN (BioAPI *BioAPI_GUI_PROGRESS_EVENT_HANDLER)
        (const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_ID UnitID,
        const BioAPI_HANDLE *BSPHandle,
        void *GUIProgressEventHandlerCtx,
        BioAPI_GUI_OPERATION Operation,
        BioAPI_GUI_SUBOPERATION Suboperation,
        BioAPI_BIR_PURPOSE Purpose,
        BioAPI_GUI_MOMENT Moment,
        uint8_t SuboperationProgress,
        const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
        const uint8_t *Text,
        BioAPI_GUI_RESPONSE *Response);
```

17.4.2 The corresponding C function pointer type in the BioSPI interface is defined in BioAPI as follows:

```
typedef BioAPI_RETURN (BioAPI *BioSPI_GUI_PROGRESS_EVENT_HANDLER)
        (const BioAPI_UUID *BSPUuid,
        BioAPI_UNIT_ID UnitID,
        const BioAPI_HANDLE *BSPHandle,
        BioAPI_GUI_OPERATION Operation,
        BioAPI_GUI_SUBOPERATION Suboperation,
        BioAPI_BIR_PURPOSE Purpose,
        BioAPI_GUI_MOMENT Moment,
        uint8_t SuboperationProgress,
        const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
        const uint8_t *Text,
        BioAPI_GUI_RESPONSE *Response);
```

17.4.3 A pair of BIP message types are related to this function: the **guiProgressEvent** notification BIP message type and the **guiProgressEvent** acknowledgement BIP message type. These two BIP message types carry a value of the following BIP message parameter ASN.1 types (respectively):

```
GUIProgressEvent-NotificationParams ::= SEQUENCE {
        guiEventSubscriptionUuid        BioAPI-UUID OPTIONAL,
        bspProductUuid                  BioAPI-UUID,
        unitID                          BioAPI-UNIT-ID,
        originalBSPHandle               BioAPI-HANDLE OPTIONAL,
        operation                       BioAPI-GUI-OPERATION,
        suboperation                    BioAPI-GUI-SUBOPERATION,
        purpose                         BioAPI-BIR-PURPOSE,
        moment                          BioAPI-GUI-MOMENT,
        suboperationProgress            UnsignedByte,
        bitmaps                         BioAPI-GUI-BITMAP-ARRAY OPTIONAL,
        text                            UTF8String OPTIONAL
}
```

and:

```
GUIProgressEvent-AcknowledgementParams ::= SEQUENCE {
        response                        BioAPI-GUI-RESPONSE
    }
```

17.4.4 The following ASN.1 types are defined to aid the specification of the behaviour of a framework, but their abstract values do not occur in any BIP message exchanged between BIP endpoints:

```
GUIProgressEventHandlerCallbackParams ::= SEQUENCE {
        guiProgressEventHandlerAddress  MemoryAddress,
        guiProgressEventHandlerContext  MemoryAddress,
        bspUuid                         BioAPI-UUID,
        unitID                          BioAPI-UNIT-ID,
        bspHandle                       BioAPI-HANDLE OPTIONAL,
        operation                       BioAPI-GUI-OPERATION,
        suboperation                    BioAPI-GUI-SUBOPERATION,
        purpose                         BioAPI-BIR-PURPOSE,
        moment                          BioAPI-GUI-MOMENT,
        suboperationProgress            UnsignedByte,
        bitmaps                         BioAPI-GUI-BITMAP-ARRAY OPTIONAL,
        text                            UTF8String OPTIONAL
    }

GUIProgressEventInfo ::= SEQUENCE {
        subscriberEndpointIRI           EndpointIRI,
        guiEventSubscriptionUuid        BioAPI-UUID OPTIONAL,
        hostingEndpointIRI              EndpointIRI,
        bspProductUuid                  BioAPI-UUID,
        unitID                          BioAPI-UNIT-ID,
        originalBSPHandle               BioAPI-HANDLE OPTIONAL,
        operation                       BioAPI-GUI-OPERATION,
        suboperation                    BioAPI-GUI-SUBOPERATION,
        purpose                         BioAPI-BIR-PURPOSE,
        moment                          BioAPI-GUI-MOMENT,
        suboperationProgress            UnsignedByte,
        bitmaps                         BioAPI-GUI-BITMAP-ARRAY OPTIONAL,
        text                            UTF8String OPTIONAL
    }
```

17.4.5 When a framework receives a call to the callback function **BioSPI_GUI_PROGRESS_EVENT_HANDLER** from a BSP, it shall perform the following actions (in order):

a) create a temporary abstract value (say, *incomingNotificationParams*) of type **GUIProgressEvent-NotificationParams** (see 17.4.3) by converting from the parameters of the function call as specified in 17.4.7;

b) search the **GUIEventRedirectors** table (see 18.12) for an entry where the component **originalBSPHandle** has the same value as the component **originalBSPHandle** of *incomingNotificationParams* and the component **guiProgressEventRedirected** has the value **TRUE**;

c) if there is no matching entry, then create a temporary abstract value (say, *eventInfo*) of type **GUIProgressEventInfo** (see 17.4.4) where:

   1) if the **AttachSessionRemoteReferences** table (see 18.9) has an entry where the component **originalBSPHandle** has the same value as the component **originalBSPHandle** of *incomingNotificationParams*, then the component **subscriberEndpointIRI** of *eventInfo* shall be set from the component **referrerEndpointIRI** of that entry; otherwise, it shall be set to the local endpoint IRI;

   2) the optional component **guiEventSubscriptionUuid** shall be absent;

   3) the component **hostingEndpointIRI** shall be set to the local endpoint IRI; and

   4) the remaining components shall be set from the components of *incomingNotificationParams* with the same names;

d) if there is one matching entry (say, *redirector*), then create a temporary abstract value (say, *eventInfo*) of type **GUIProgressEventInfo** (see 17.4.4) where:

1) the component **subscriberEndpointIRI** shall be set from the component **subscriberEndpointIRI** of *redirector*;

2) the optional component **guiEventSubscriptionUuid** shall be present and shall be set from the component **guiEventSubscriptionUuid** of *redirector*;

3) the component **hostingEndpointIRI** shall be set to the local endpoint IRI;

4) the optional component **originalBSPHandle** shall be absent; and

5) the remaining components shall be set from the components of *incomingNotificationParams* with the same names;

e) notify a GUI progress event based on *eventInfo* to a subscriber (either a GUI progress event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 32;

NOTE – If the subscriber of the GUI progress event is a master endpoint (remote application), then the presence of a large image in the notification may cause a delay in the remote display. It is therefore desirable for a BSP to avoid sending large images.

f) if *incomingReturnValue* is not 0, then return that value to the BSP, skipping the remaining actions;

g) set the output parameters of the call to the callback function **BioSPI_GUI_PROGRESS_EVENT_HANDLER** by converting from *incomingAcknowledgementParams* as specified in 17.4.9; and

h) return the value 0 to the BSP.

17.4.6 When a framework receives (see 13.9) a **guiProgressEvent** notification BIP message from a slave endpoint, it shall perform the following actions (in order):

a) let *incomingNotificationParams* be the parameter value (of type **GUIProgressEvent-NotificationParams** – see 17.4.3) of the notification BIP message;

b) create a temporary abstract value (say, *eventInfo*) of type **GUIProgressEventInfo** (see 17.4.4) where:

1) the components **subscriberEndpointIRI** shall be set to the local endpoint IRI;

2) the component **hostingEndpointIRI** shall be set from the component **slaveEndpointIRI** of the notification BIP message; and

3) the remaining components shall be set from the components of *incomingNotificationParams* with the same names;

c) notify a GUI progress event based on *eventInfo* to a subscriber (either a GUI progress event handler of the local application or a master endpoint) and determine the acknowledgement parameter value (say, *incomingAcknowledgementParams*) and the acknowledgement return value (say, *incomingReturnValue*) as specified in clause 32;

d) if *incomingReturnValue* is not 0, then create and send a corresponding **guiProgressEvent** acknowledgement BIP message (see 13.5) with the return value set to that value, skipping the remaining actions;

e) create and send a corresponding **guiProgressEvent** acknowledgement BIP message (see 13.5) with the parameter value set to *incomingAcknowledgementParams* and the return value set to 0.

17.4.7 Conversion from the parameters of the C callback function **BioSPI_GUI_PROGRESS_EVENT_HANDLER** to the ASN.1 type **GUIProgressEvent-NotificationParams** (see 17.4.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 132.

Table 132 – Mapping from the parameters of the callback function BioSPI_GUI_PROGRESS_EVENT_HANDLER to the ASN.1 type GUIProgressEvent-NotificationParams

Function parameter	Component of the ASN.1 type	References
none	guiEventSubscriptionUuid	17.4.8
BSPUuid	bspProductUuid	clause 19 in conjunction with 15.58
UnitID	unitID	15.55
BSPHandle	originalBSPHandle	clause 19 in conjunction with 15.42
Operation	operation	15.39
Suboperation	suboperation	15.41
Purpose	purpose	15.14
Moment	moment	15.37
SuboperationProgress	suboperationProgress	15.1.3
Bitmaps	bitmaps	clause 19 in conjunction with 15.35
Text	text	15.2
Response	*none*	clause 22

17.4.8 The optional component **guiEventSubscriptionUuid** shall be absent.

17.4.9 Conversion from the ASN.1 type **GUIProgressEvent-AcknowledgementParams** (see 17.4.3) to the parameters of the C callback function **BioSPI_GUI_PROGRESS_EVENT_HANDLER** shall be done by converting from individual ASN.1 components to function parameters in accordance with Table 133.

Table 133 – Mapping from the ASN.1 type GUIProgressEvent-AcknowledgementParams to the parameters of the callback function BioSPI_GUI_PROGRESS_EVENT_HANDLER

Component of the ASN.1 type	Function parameter	References
response	Response	clause 20 in conjunction with 15.40

17.4.10 Conversion from the ASN.1 type **GUIProgressEventHandlerCallbackParams** (see 17.4.4) to the parameters of the C callback function **BioAPI_GUI_PROGRESS_EVENT_HANDLER** shall be done by converting from individual ASN.1 components to function parameters with Table 134.

Table 134 – Mapping from the ASN.1 type GUIProgressEventHandlerCallbackParams to the parameters of the callback function BioAPI_GUI_PROGRESS_EVENT_HANDLER

Component of the ASN.1 type	Function parameter	References
guiProgressEventHandlerAddress	*none*	17.4.11
guiProgressEventHandlerContext	GUIProgressEventHandlerCtx	15.1.7
bspUuid	BSPUuid	clause 19 in conjunction with 15.58
unitID	UnitID	15.55
bspHandle	BSPHandle	clause 19 in conjunction with 15.42
operation	Operation	15.39
suboperation	Suboperation	15.41
purpose	Purpose	15.14
moment	Moment	15.37
suboperationProgress	SuboperationProgress	15.1.3
bitmaps	Bitmaps	clause 19 in conjunction with 15.35
text	Text	15.2
none	Response	clause 22

17.4.11 The component **guiProgressEventHandlerAddress** corresponds to the address of the function rather than to a parameter of the function. Conversion shall be done by applying 15.1.7.

17.4.12 Conversion from the parameters of the C callback function **BioAPI_GUI_PROGRESS_EVENT_HANDLER** to the ASN.1 type **GUIProgressEvent-AcknowledgementParams** (see 17.4.3) shall be done by converting from individual function parameters to ASN.1 components in accordance with Table 135.

Table 135 – Mapping from the parameters of the callback function BioAPI_GUI_PROGRESS_EVENT_HANDLER to the ASN.1 type GUIProgressEvent-AcknowledgementParams

Function parameter	Component of the ASN.1 type	References
Response	response	clause 20 in conjunction with 15.40

18 Conceptual tables

This clause specifies ASN.1 type definitions for a series of conceptual tables. These ASN.1 types are defined to aid the specification of the behaviour of a framework, but their abstract values do not occur in any BIP message exchanged between BIP endpoints, and therefore are never encoded. A conforming master BIP endpoint is required to support all the tables specified in this clause. A conforming slave BIP endpoint is required to support all the tables specified in this clause for slave endpoints. A BIP endpoint can use any suitable representation for the conceptual tables and is not required to support serialization of their content.

NOTE – A BIP endpoint implementation may choose to support such serialization for administrative or debugging purposes, but this is not required. An example of this capability would be a management interface that supports a query of the current content of the conceptual tables.

18.1 The MasterEndpoints conceptual table

This conceptual table shall be present in all slave endpoints and is defined in ASN.1 as follows:

```
MasterEndpoints ::= SET OF endpoint MasterEndpoint
        MasterEndpoint ::= SEQUENCE {
                masterEndpointIRI               EndpointIRI
        }
```

18.1.1 General

An entry of this table represents a master endpoint that is a BIP endpoint linked to the local endpoint.

18.1.2 Components

18.1.2.1 The component **masterEndpointIRI** shall contain the endpoint IRI of the master endpoint. This shall be different from the local endpoint IRI.

18.1.2.2 There shall not be two or more entries with the same value of the component **masterEndpointIRI**.

18.1.3 Entry deletion

18.1.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **MasterEndpoints** table is deleted.

18.1.3.2 Let *masterEndpointIRI* be the value of the component **masterEndpointIRI** of the entry being deleted.

18.1.3.3 For each entry of the **RunningBSPRemoteReferences** table (see 18.6) where the component **referrerEndpointIRI** has the value *masterEndpointIRI*, the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *bspUnloadCallParams*) of type **BSPUnloadCallParams** (see 16.10.3) where:

   1) the component **bspUuid** shall be set from the component **bspProductUuid** of the entry; and

   2) the components **unitEventHandlerAddress** and **unitEventHandlerContext** shall be set to 0;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_BSPUnload** (see 16.10), where the parameters of the function call shall be set by converting from *bspUnloadCallParams* as specified in 16.10.3;

c) delete the entry of the **RunningBSPRemoteReferences** table (subclause 18.6.3 applies).

18.1.3.4 For each entry of the **UnitEventNotificationDisablers** table (see 18.7) where the component **referrerEndpointIRI** has the value **masterEndpointIRI**, the framework shall delete the entry of the **UnitEventNotificationDisablers** table (subclause 18.7.3 applies).

18.1.3.5 For each entry of the **GUIEventRemoteSubscriptions** table (see 18.11) where the component **subscriberEndpointIRI** has the value *masterEndpointIRI*, the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *unsubscribeFromGUIEventsCallParams*) of type **UnsubscribeFromGUIEventsCallParams** (see 16.23.3) where:

1) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of the entry (presence and value);

2) if the optional component **originalBSPHandle** of the entry is absent, then the optional component **bspUuid** of *unsubscribeFromGUIEventsCallParams* shall be set from the component **bspProductUuid** of the entry; otherwise, it shall be absent;

3) the optional component **bspHandle** shall be set from the optional component **originalBSPHandle** of the entry (presence and value);

4) if the component **guiSelectEventSubscribed** of the entry has the value **TRUE**, then the component **guiSelectEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

5) if the component **guiStateEventSubscribed** of the entry has the value **TRUE**, then the component **guiStateEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

6) if the component **guiProgressEventSubscribed** of the entry has the value **TRUE**, then the component **guiProgressEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0; and

7) the components **guiSelectEventHandlerContext**, **guiStateEventHandlerContext**, and **guiProgressEventHandlerContext** shall be set to 0;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnsubscribeFromGUIEvents** (see 16.23), where the parameters of the function call shall be set by converting from *unsubscribeFromGUIEventsCallParams* as specified in 16.23.5;

c) delete the entry of the **GUIEventRemoteSubscriptions** table (subclause 18.11.3 applies).

18.1.3.6 For each entry of the **GUIEventRedirectors** table (see 18.12) where the component **subscriberEndpointIRI** has the value *masterEndpointIRI*, the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *unredirectCallParams*) of type **UnredirectGUIEvents-RequestParams** (see 16.29.2), where all the components shall be set from the components of the entry with the same names;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnredirectGUIEvents** (see 16.29), where the parameters of the function call shall be set by converting from *unredirectCallParams* as specified in 16.29.5;

c) delete the entry of the **GUIEventRedirectors** table (subclause 18.12.3 applies).

18.1.4 Life cycle

18.1.4.1 An entry can be added to the **MasterEndpoints** table:

– when a framework receives an **addMaster** request BIP message from a BIP endpoint (see 16.4.2).

18.1.4.2 An entry of the **MasterEndpoints** table can be deleted:

a) when a framework receives a call to the function **BioAPI_Terminate** from the local application (see 16.3); and

b) when a framework receives a **deleteMaster** request BIP message from a BIP endpoint (see 16.5.2).

18.2 The VisibleEndpoints conceptual table

This conceptual table shall be present in all BIP endpoints and is defined in ASN.1 as follows:

```
VisibleEndpoints ::= SET OF endpoint VisibleEndpoint

VisibleEndpoint ::= BioAPI-FRAMEWORK-SCHEMA
```

18.2.1 General

18.2.1.1 An entry of this table represents a visible endpoint, which is a BIP endpoint whose components (framework, BSPs, and BFPs) are visible to the local application.

18.2.1.2 This table has one entry for the local endpoint and one entry for each slave endpoint of the local endpoint (zero or more). Each entry contains a copy of the framework schema present in the component registry of the (local or slave) BIP endpoint.

18.2.2 Components

18.2.2.1 The component **hostingEndpointIRI** shall contain the endpoint IRI of the visible endpoint. The visible endpoint shall be either the local endpoint or a slave endpoint.

18.2.2.2 The other components shall contain a copy of the attributes of the framework schema in the component registry of the BIP endpoint.

18.2.2.3 There shall not be two or more entries with the same value of the component **hostingEndpointIRI**.

18.2.3 Entry deletion

18.2.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **VisibleEndpoints** table is deleted.

18.2.3.2 Let *hostingEndpointIRI* be the value of the component **hostingEndpointIRI** of the entry being deleted.

18.2.3.3 For each entry of the **VisibleBSPRegistrations** table (see 18.3) where the component **hostingEndpointIRI** has the value *hostingEndpointIRI*, the framework shall delete the entry (subclause 18.3.3 applies).

18.2.3.4 For each entry of the **VisibleBFPRegistrations** table (see 18.4) where the component **hostingEndpointIRI** has the value *hostingEndpointIRI*, the framework shall delete the entry (subclause 18.4.3 applies).

18.2.4 Life cycle

18.2.4.1 An entry can be added to the **VisibleEndpoints** table:

a) when a framework receives a call to the function **BioAPI_Init** or **BioAPI_InitEndpoint** from the local application (see 16.1); and

b) when a framework receives a call to the function **BioAPI_LinkToEndpoint** from the local application (see 16.4).

18.2.4.2 An entry of the **VisibleEndpoints** table can be deleted:

a) when a framework receives a **masterDeletionEvent** notification BIP message from a slave endpoint (see 16.3.2); and

b) when a framework receives a call to the function **BioAPI_UnlinkFromEndpoint** from the local application (see 16.5).

18.3 The VisibleBSPRegistrations conceptual table

This conceptual table shall be present in all BIP endpoints and is defined in ASN.1 as follows:

```
VisibleBSPRegistrations ::= SET OF
                registration VisibleBSPRegistration

VisibleBSPRegistration ::= BioAPI-BSP-SCHEMA
```

18.3.1 General

18.3.1.1 An entry of this table represents a BSP registered in the component registry of a visible endpoint.

18.3.1.2 This table has one entry for each BSP registered in the local component registry and one entry for each BSP registered in the component registry of each slave endpoint of the local endpoint. Each entry contains a copy of a BSP schema present in the component registry of a (local or slave) BIP endpoint.

18.3.1.3 If a BSP is registered in the component registries of two or more visible endpoints, the table will have a separate entry for each BSP/endpoint pair, and those entries will differ at least in the value of their component **hostingEndpointIRI**.

18.3.1.4 This table also supports translation between BSP product UUIDs and BSP access UUIDs.

NOTE – A BSP product UUID identifies a BSP as a software product, is assigned by the vendor of the BSP, and is included in the registered BSP schema. The BSP product UUID usually does not vary across multiple registrations of the same BSP in different component registries. A BSP access UUID is dynamically generated by the framework, is assigned to each BSP represented in the table, and is not visible outside the local endpoint. When a framework has one or more slave endpoints, the same BSP may happen to be registered in two or more visible endpoints, and therefore there may be two or more entries containing the same BSP product UUID in the table. In this case, the use of a BSP product UUID in a call to **BioAPI_BSPLoad** will not be sufficient to unambiguously identify both the BSP to be loaded and the BIP endpoint where the BSP is to be loaded. The BSP access UUID, instead, can be used in any call to **BioAPI_BSPLoad**. On the other hand, the BSP product UUID can be used in a call to **BioAPI_BSPLoad** so long as the BSP is represented exactly once in the table, because in this case the BSP product UUID is sufficient to identify both the BSP to be loaded and the BIP endpoint where the BSP is to be loaded.

18.3.2 Components

18.3.2.1 The component **hostingEndpointIRI** shall contain the endpoint IRI of the BIP endpoint whose component registry contains the BSP schema of the registered BSP. The BIP endpoint shall be either the local endpoint or a slave endpoint.

18.3.2.2 The component **bspAccessUuid** shall contain the BSP access UUID dynamically generated by the framework to identify the registered BSP.

18.3.2.3 The other components shall contain a copy of the attributes of the BSP schema in the component registry of the BIP endpoint.

18.3.2.4 There shall not be two or more entries with the same values of both components **hostingEndpointIRI** and **bspProductUuid**.

18.3.2.5 There shall not be two or more entries with the same value of the component **bspAccessUuid**.

18.3.2.6 The value of the component **bspAccessUuid** in an entry shall be different from the value of the component **bspProductUuid** in either that entry or any other entry.

18.3.3 Entry deletion

18.3.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **VisibleBSPRegistrations** table is deleted.

18.3.3.2 Let *hostingEndpointIRI* be the value of the component **hostingEndpointIRI** and *bspProductUuid* the value of the component **bspProductUuid** of the entry being deleted.

18.3.3.3 For each entry of the **RunningBSPLocalReferences** table (see 18.5) where the component **hostingEndpointIRI** has the value *hostingEndpointIRI* and the component **bspProductUuid** has the value *bspProductUuid*, the framework shall perform the following actions (in order):

a) if *hostingEndpointIRI* is not the local endpoint IRI, then delete the entry of the **RunningBSPLocalReferences** table (subclause 18.5.3 applies), skipping the remaining actions;

b) create a temporary abstract value (say, *bspUnloadCallParams*) of type **BSPUnloadCallParams** (see 16.10.3) where:

   1) the component **bspUuid** shall be set to *bspProductUuid*; and

   2) the components **unitEventHandlerAddress** and **unitEventHandlerContext** shall be set to 0;

c) make an internal BioAPI function call (see 13.10) to the function **BioAPI_BSPUnload** (see 16.10), where the parameters of the function call shall be set by converting from *bspUnloadCallParams* as specified in 16.10.3;

d) delete the entry of the **RunningBSPLocalReferences** table (subclause 18.5.3 applies).

18.3.3.4 If *hostingEndpointIRI* is the local endpoint IRI, then, for each entry of the **RunningBSPRemoteReferences** table (see 18.6) where the component **bspProductUuid** has the value *bspProductUuid*, the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *bspUnloadCallParams*) of type **BSPUnloadCallParams** (see 16.10.3) where:

1) the component **bspUuid** shall be set to *bspProductUuid*; and

2) the components **unitEventHandlerAddress** and **unitEventHandlerContext** shall be set to 0;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_BSPUnload** (see 16.10), where the parameters of the function call shall be set by converting from *bspUnloadCallParams* as specified in 16.10.3;

c) delete the entry of the **RunningBSPRemoteReferences** table (subclause 18.6.3 applies).

18.3.3.5 For each entry of the **GUIEventLocalSubscriptions** table (see 18.10) where the component **hostingEndpointIRI** has the value *hostingEndpointIRI* and the component **bspProductUuid** has the value *bspProductUuid*, the framework shall perform the following actions (in order):

a) if *hostingEndpointIRI* is not the local endpoint IRI, then delete the entry of the **GUIEventLocalSubscriptions** table (subclause 18.10.3 applies), skipping the remaining actions;

b) create a temporary abstract value (say, *unsubscribeFromGUIEventsCallParams*) of type **UnsubscribeFromGUIEventsCallParams** (see 16.23.3), where:

1) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of the entry (presence and value);

2) if the optional component **originalBSPHandle** of the entry is absent, then the optional component **bspUuid** of *unsubscribeFromGUIEventsCallParams* shall be set from the component **bspProductUuid** of the entry; otherwise, it shall be absent;

3) the optional component **bspHandle** shall be set from the optional component **originalBSPHandle** of the entry (presence and value);

4) if the component **guiSelectEventHandlerAddress** of the entry has a value different from 0, then the component **guiSelectEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

5) if the component **guiStateEventHandlerAddress** of the entry has a value different from 0, then the component **guiStateEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

6) if the component **guiProgressEventHandlerAddress** of the entry has a value different from 0, then the component **guiProgressEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0; and

7) the components **guiSelectEventHandlerContext**, **guiStateEventHandlerContext**, and **guiProgressEventHandlerContext** shall be set to 0;

c) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnsubscribeFromGUIEvents** (see 16.23), where the parameters of the function call shall be set by converting from *unsubscribeFromGUIEventsCallParams* as specified in 16.23.6;

d) delete the entry of the **GUIEventLocalSubscriptions** table (subclause 18.10.3 applies);

18.3.3.6 If *hostingEndpointIRI* is the local endpoint IRI, then, for each entry of the **GUIEventRemoteSubscriptions** table (see 18.11) where the component **bspProductUuid** has the value *bspProductUuid*, the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *unsubscribeFromGUIEventsCallParams*) of type **UnsubscribeFromGUIEventsCallParams** (see 16.23.3) where:

1) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of the entry (presence and value);

2) if the optional component **originalBSPHandle** of the entry is absent, then the optional component **bspUuid** of *unsubscribeFromGUIEventsCallParams* shall be set from the component **bspProductUuid** of the entry; otherwise, it shall be absent;

3) the optional component **bspHandle** shall be set from the optional component **originalBSPHandle** of the entry (presence and value);

4) if the component **guiSelectEventSubscribed** of the entry has the value **TRUE**, then the component **guiSelectEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

5) if the component **guiStateEventSubscribed** of the entry has the value **TRUE**, then the component **guiStateEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

6) if the component **guiProgressEventSubscribed** of the entry has the value **TRUE**, then the component **guiProgressEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0; and

7) the components **guiSelectEventHandlerContext**, **guiStateEventHandlerContext**, and **guiProgressEventHandlerContext** shall be set to 0;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnsubscribeFromGUIEvents** (see 16.23), where the parameters of the function call shall be set by converting from *unsubscribeFromGUIEventsCallParams* as specified in 16.23.6;

c) delete the entry of the **GUIEventRemoteSubscriptions** table (subclause 18.11.3 applies).

18.3.4 Life cycle

18.3.4.1 An entry can be added to the **VisibleBSPRegistrations** table:

a) when a framework receives a call to the function **BioAPI_Init** or **BioAPI_InitEndpoint** from the local application (see 16.1);

b) when a framework receives a call to the function **BioAPI_LinkToEndpoint** from the local application (see 16.4);

c) when a framework receives a call to the function **BioAPI_RegisterBSP** from the local application (see 16.59);

d) when a framework receives a **registerBSP** request BIP message from a master endpoint (see 16.59.2); and

e) when a framework receives a **bspRegistrationEvent** notification BIP message from a slave endpoint (see 16.59.2).

18.3.4.2 An entry of the **VisibleBSPRegistrations** table can be deleted:

a) when a framework receives a call to the function **BioAPI_RegisterBSP** from the local application (see 16.59);

b) when a framework receives a **registerBSP** request BIP message from a master endpoint (see 16.59.2);

c) when a framework receives a **bspRegistrationEvent** notification BIP message from a slave endpoint (see 16.59.2);

d) when a framework receives a call to the function **BioAPI_UnregisterBSP** from the local application (see 16.60);

e) when a framework receives an **unregisterBSP** request BIP message from a master endpoint (see 16.60.2);

f) when a framework receives a **bspUnregistrationEvent** notification BIP message from a slave endpoint (see 16.60.2); and

g) when an entry of the **VisibleEndpoints** table is deleted (see 18.2.4.2).

18.4 The VisibleBFPRegistrations conceptual table

This conceptual table shall be present in all BIP endpoints and is defined in ASN.1 as follows:

```
VisibleBFPRegistrations ::= SET OF
        registration VisibleBFPRegistration

VisibleBFPRegistration ::= BioAPI-BFP-SCHEMA
```

18.4.1 General

18.4.1.1 An entry of this table represents a BFP as registered in the component registry of a visible endpoint, and thus available for use by the local application through a registered BSP within the same BIP endpoint.

18.4.1.2 This table has one entry for each BFP registered in the local component registry and one entry for each BFP registered in the component registry of each slave endpoint of the local endpoint. Each entry contains a copy of a BFP schema present in the component registry of a (local or slave) BIP endpoint.

18.4.1.3 If a BFP is registered in the component registries of two or more visible endpoints, the table will have a separate entry for each BFP/endpoint pair, and those entries will differ at least in the value of their component **hostingEndpointIRI**.

18.4.2 Components

18.4.2.1 The component **hostingEndpointIRI** shall contain the endpoint IRI of the BIP endpoint whose component registry contains the BFP schema of the registered BFP. The BIP endpoint shall be either the local endpoint or a slave endpoint.

18.4.2.2 The other components shall contain a copy of the attributes of the BFP schema in the component registry of the BIP endpoint.

18.4.2.3 There shall not be two or more entries with the same values of both components **hostingEndpointIRI** and **bfpProductUuid**.

18.4.3 Entry deletion

18.4.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **VisibleBFPRegistrations** table is deleted.

18.4.3.2 No additional action is required.

18.4.4 Life cycle

18.4.4.1 An entry can be added to the **VisibleBFPRegistrations** table:

a) when a framework receives a call to the function **BioAPI_Init** or **BioAPI_InitEndpoint** from the local application (see 16.1);

b) when a framework receives a call to the function **BioAPI_LinkToEndpoint** from the local application (see 16.4);

c) when a framework receives a call to the function **BioAPI_RegisterBFP** from the local application (see 16.61);

d) when a framework receives a **registerBFP** request BIP message from a master endpoint (see 16.61.2); and

e) when a framework receives a **bfpRegistrationEvent** notification BIP message from a slave endpoint (see 16.61.2).

18.4.4.2 An entry of the **VisibleBFPRegistrations** table can be deleted:

a) when a framework receives a call to the function **BioAPI_RegisterBFP** from the local application (see 16.61);

b) when a framework receives a **registerBFP** request BIP message from a master endpoint (see 16.61.2);

c) when a framework receives a **bfpRegistrationEvent** notification BIP message from a slave endpoint (see 16.61.2);

d) when a framework receives a call to the function **BioAPI_UnregisterBFP** from the local application (see 16.62);

e) when a framework receives an **unregisterBFP** request BIP message from a master endpoint (see 16.62.2);

f) when a framework receives a **bfpUnregistrationEvent** notification BIP message from a slave endpoint (see 16.62.2); and

g) when an entry of the **VisibleEndpoints** table is deleted (see 18.2.4.2).

18.5 The RunningBSPLocalReferences conceptual table

This conceptual table shall be present in all BIP endpoints and is defined in ASN.1 as follows:

```
RunningBSPLocalReferences ::= SET OF
        reference RunningBSPLocalReference

RunningBSPLocalReference ::= SEQUENCE {
        hostingEndpointIRI          EndpointIRI,
        bspProductUuid              BioAPI-UUID,
        useBSPAccessUuid            BOOLEAN,
        unitEventHandlerAddress     MemoryAddress,
        unitEventHandlerContext     MemoryAddress
    }
```

18.5.1 General

18.5.1.1 An entry of this table represents a running BSP local reference, consisting of:

a) a reference (established by a call to **BioAPI_BSPLoad**) held by the local application to a BSP running in either the local endpoint or a slave endpoint; and

b) (optionally) an obligation for the framework to make a callback to a unit event handler of the local application on certain incoming unit event notifications (either a unit event notification callback from the BSP or a unit event notification BIP message related to the BSP).

18.5.1.2 An incoming call to **BioAPI_BSPLoad** can create a new running BSP local reference entry even if the parameters of the call are the same as those of a previous incoming call to **BioAPI_BSPLoad**. An incoming call to **BioAPI_BSPUnload** whose parameters match an existing running BSP local reference entry can delete that entry. In case of multiple matches (multiple identical entries), any one of the matching entries can be deleted.

18.5.1.3 A running BSP local reference can use either the BSP product UUID of a BSP, or the BSP access UUID dynamically assigned to the BSP by the framework. The local application can obtain the BSP access UUID of a BSP by calling **BioAPI_EnumBSPs**.

18.5.1.4 Normally, a running BSP is not unloaded as long as the local application (if any) holds a running BSP local reference to it, or as long as a master endpoint holds a running BSP remote reference to it.

18.5.2 Components

18.5.2.1 The component **hostingEndpointIRI** shall contain the endpoint IRI of the BIP endpoint containing the registered BSP referenced in the running BSP local reference. The BIP endpoint shall be either the local endpoint or a slave endpoint.

18.5.2.2 The component **bspProductUuid** shall contain the BSP product UUID of the registered BSP.

18.5.2.3 The component **useBSPAccessUuid** shall indicate whether the local application provided the BSP access UUID (rather than the BSP product UUID) to identify the registered BSP in the **BioAPI_BSPLoad** call (see 16.9) that caused the addition of the entry.

NOTE – A subsequent **BioAPI_BSPUnload** (see 16.10) call has to provide the same UUID (either the BSP access UUID or the BSP product UUID) in order to match this running BSP local reference. In addition, unit event notification callbacks generated using this running BSP local reference will provide the same UUID as input to the unit event handler of the local application.

18.5.2.4 The component **unitEventHandlerAddress** shall contain the callback address of a unit event handler of the local application. A value different from 0 signifies an obligation for the framework to make a callback to a unit event handler using that callback address.

18.5.2.5 The component **unitEventHandlerContext** shall contain the context address to be passed as input to the unit event handler of the local application.

18.5.2.6 There may be multiple entries with the same values of one or more (or even all) of their components.

18.5.3 Entry deletion

18.5.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **RunningBSPLocalReferences** table is deleted.

18.5.3.2 Let *hostingEndpointIRI* be the value of the component **hostingEndpointIRI** and *bspProductUuid* the value of the component **bspProductUuid** of the entry being deleted.

18.5.3.3 If the entry being deleted is the only entry in the **RunningBSPLocalReferences** table where the component **hostingEndpointIRI** has the value *hostingEndpointIRI* and the component **bspProductUuid** has the value *bspProductUuid*, then, for each entry of the **AttachSessionLocalReferences** table (see 18.8) where the component **hostingEndpointIRI** has the value *hostingEndpointIRI* and the component **bspProductUuid** has the value *bspProductUuid*, the framework shall perform the following actions (in order):

a) if *hostingEndpointIRI* is not the local endpoint IRI, then delete the entry of the **AttachSessionLocalReferences** table (subclause 18.8.3 applies), skipping the remaining actions;

b) create a temporary abstract value (say, *bspDetachCallParams*) of type **BSPDetach-RequestParams** (see 16.14.2) where the component **originalBSPHandle** shall be set from the component **originalBSPHandle** of the entry;

c) make an internal BioAPI function call (see 13.10) to the function **BioAPI_BSPDetach** (see 16.14), where the parameters of the function call shall be set by converting from *bspDetachCallParams* as specified in 16.14.5;

d) delete the entry of the **AttachSessionLocalReferences** table (subclause 18.8.3 applies).

18.5.4 Life cycle

18.5.4.1 An entry can be added to the **RunningBSPLocalReferences** table:

– when a framework receives a call to the function **BioAPI_BSPLoad** from the local application (see 16.9).

18.5.4.2 An entry of the **RunningBSPLocalReferences** table can be deleted:

a) when a framework receives a call to the function **BioAPI_BSPUnload** from the local application (see 16.10); and

b) when an entry of the **VisibleBSPRegistrations** table is deleted (see 18.3.4.2).

18.6 The RunningBSPRemoteReferences conceptual table

This conceptual table shall be present in all slave endpoints and is defined in ASN.1 as follows:

```
RunningBSPRemoteReferences ::= SET OF
                reference RunningBSPRemoteReference

RunningBSPRemoteReference ::= SEQUENCE {
                referrerEndpointIRI         EndpointIRI,
                bspProductUuid              BioAPI-UUID,
                unitEventSubscription       BOOLEAN
        }
```

18.6.1 General

18.6.1.1 An entry of this table represents a running BSP remote reference, consisting of:

a) a reference (established by a **bspLoad** request BIP message) held by a master endpoint to a BSP running in the local endpoint; and

b) (optionally) an obligation for the framework to send a **unitEvent** notification BIP message to the master endpoint on certain incoming unit event notification callbacks from the BSP.

18.6.1.2 An incoming **bspLoad** request BIP message can create a new running BSP remote reference entry even if the parameter value of the BIP message is the same as that of a previous incoming **bspLoad** request BIP message. An incoming **bspUnload** request BIP message whose parameter value (in addition to the endpoint IRI of the sending BIP endpoint) matches an existing running BSP remote reference entry can delete that entry. In case of multiple matches (multiple identical entries), any one of the matching entries can be deleted.

18.6.1.3 Normally, a running BSP is not unloaded as long as the local application (if any) holds a running BSP local reference to it, or as long as a master endpoint holds a running BSP remote reference to it.

18.6.2 Components

18.6.2.1 The component **referrerEndpointIRI** shall contain the endpoint IRI of the master endpoint that holds the running BSP remote reference. This shall not be the local endpoint.

18.6.2.2 The component **bspProductUuid** shall contain the BSP product UUID of the registered BSP.

18.6.2.3 The component **unitEventSubscription** shall indicate whether there is an obligation for the framework to send **unitEvent** notification BIP messages to the master endpoint.

18.6.2.4 There may be multiple entries with the same values of one or more (or even all) of their components.

18.6.3 Entry deletion

18.6.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **RunningBSPRemoteReferences** table is deleted.

18.6.3.2 Let *referrerEndpointIRI* be the value of the component **referrerEndpointIRI** and let *bspProductUuid* be the value of the component **bspProductUuid** of the entry being deleted.

18.6.3.3 If the entry being deleted is the only entry in the **RunningBSPRemoteReferences** table where the component **referrerEndpointIRI** has the value *referrerEndpointIRI* and the component **bspProductUuid** has the value *bspProductUuid*, then, for each entry of the **AttachSessionRemoteReferences** table (see 18.9) where the component **referrerEndpointIRI** has the value *referrerEndpointIRI* and the component **bspProductUuid** has the value *bspProductUuid*, the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *bspDetachCallParams*) of type **BSPDetach-RequestParams** (see 16.14.2) where the component **originalBSPHandle** shall be set from the component **originalBSPHandle** of the entry;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_BSPDetach** (see 16.14), where the parameters of the function call shall be set by converting from *bspDetachCallParams* as specified in 16.14.5;

c) delete the entry of the **AttachSessionRemoteReferences** table (subclause 18.9.3 applies).

18.6.4 Life cycle

18.6.4.1 An entry can be added to the **RunningBSPRemoteReferences** table:

– when a framework receives a **bspLoad** request BIP message from a master endpoint (see 16.9.2).

18.6.4.2 An entry of the **RunningBSPRemoteReferences** table can be deleted:

a) when a framework receives a **bspUnload** request BIP message from a master endpoint (see 16.10.2);

b) when an entry of the **MasterEndpoints** table is deleted (see 18.1.4.2); and

c) when an entry of the **VisibleBSPRegistrations** table is deleted (see 18.3.4.2).

18.7 The UnitEventNotificationDisablers conceptual table

This conceptual table shall be present in all endpoints and is defined in ASN.1 as follows:

```
UnitEventNotificationDisablers ::= SET OF
                disabler UnitEventNotificationDisabler

UnitEventNotificationDisabler ::= SEQUENCE {
                referrerEndpointIRI          EndpointIRI,
                bspProductUuid               BioAPI-UUID,
                unitEventTypes               BioAPI-UNIT-EVENT-TYPE-MASK
        }
```

18.7.1 General

18.7.1.1 An entry of this table represents an event notification disabler, consisting in a prohibition for the framework to either send **unitEvent** notification BIP messages to a specific master endpoint, or call any unit event handler of the local application (depending on the value of **referrerEndpointIRI**), for a specific BSP and for certain types of events.

18.7.1.2 Entries are added to this table as a result of an incoming call to **BioAPI_EnableEventNotifications** or an incoming **enableEventNotifications** request BIP message. However, the same incoming call or request BIP message may also cause an entry of the table to be updated or deleted.

18.7.1.3 Although each entry of this table contains the product UUID of a BSP, the addition, updating, and deletion of the entries of this table are completely independent of BSPs being loaded and unloaded.

18.7.2 Components

18.7.2.1 The component **referrerEndpointIRI** shall contain either the endpoint IRI of the local endpoint or the endpoint IRI of a master endpoint.

18.7.2.2 The component **bspProductUuid** shall contain the BSP product UUID of a registered BSP for which one or more types of unit event notifications are disabled.

18.7.2.3 The component **unitEventTypes** shall designate one or more types of unit event notifications which are disabled. This component shall never have a value indicating that all unit event notifications are enabled.

18.7.2.4 There shall not be two or more entries with the same values of the components **referrerEndpointIRI** and **bspProductUuid**.

18.7.3 Entry deletion

18.7.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **UnitEventNotificationDisablers** table is deleted.

18.7.3.2 No additional action is required.

18.7.4 Life cycle

18.7.4.1 An entry can be added to the **UnitEventNotificationDisablers** table:

a) when the framework receives an **enableEventNotifications** request BIP message from a master endpoint (see 16.16.5); and

b) when the framework receives a call to the function **BioAPI_EnableEventNotifications** from the local application (see 16.16.4).

18.7.4.2 An entry of the **UnitEventNotificationDisablers** table can be deleted:

a) when the framework receives an **enableEventNotifications** request BIP message from a master endpoint (see 16.16.5);

b) when the framework receives a call to the function **BioAPI_EnableEventNotifications** from the local application (see 16.16.4); and

when an entry of the **MasterEndpoints** table is deleted (see 18.1.3).

18.8 The AttachSessionLocalReferences conceptual table

This conceptual table shall be present in all BIP endpoints and is defined in ASN.1 as follows:

```
AttachSessionLocalReferences ::= SET OF
                reference AttachSessionLocalReference

AttachSessionLocalReference ::= SEQUENCE {
                hostingEndpointIRI          EndpointIRI,
                bspProductUuid              BioAPI-UUID,
                useBSPAccessUuid            BOOLEAN,
                originalBSPHandle           BioAPI-HANDLE,
                localBSPHandle              BioAPI-HANDLE
        }
```

18.8.1 General

18.8.1.1 An entry of this table represents an attach session local reference, that is a reference (established by a call to **BioAPI_BSPAttach**) held by the local application to an attach session of a BSP running in either the local endpoint or a slave endpoint.

18.8.1.2 For a BSP running in the local endpoint, an original BSP handle and a local BSP handle are generated by the framework (see 16.13). For a BSP running in a slave endpoint, an original BSP handle is generated within the slave endpoint on reception of a **bspAttach** request BIP message from the framework (see 16.13), and a local BSP handle is generated by the framework after receiving the corresponding **bspAttach** response BIP message containing the original BSP handle (see 16.13).

18.8.1.3 This table also supports translation between local BSP handles and original BSP handles.

NOTE – This mechanism has been introduced because a BSP handle can only be guaranteed to be unique within the framework that generates it. If the original BSP handles generated by a slave framework were returned to the local application, collisions might occur with other original BSP handles generated by other slave frameworks or by the local framework. For a BSP running in the local endpoint, the framework first generates an original BSP handle without regard of the existing local BSP handles, and then assigns a local BSP handle that may be different from the original BSP handle.

18.8.2 Components

18.8.2.1 The component **hostingEndpointIRI** shall contain the endpoint IRI of the BIP endpoint containing the registered BSP referenced in the attach session local reference. The BIP endpoint shall be either the local endpoint or a slave endpoint.

18.8.2.2 The component **bspProductUuid** contains the BSP product UUID of the registered BSP.

18.8.2.3 The component **useBSPAccessUuid** shall indicate whether the local application provided the BSP access UUID (rather than the BSP product UUID) to identify the registered BSP in the **BioAPI_BSPAttach** call (see 16.13) that caused the addition of the entry.

18.8.2.4 The component **originalBSPHandle** shall contain the original BSP handle.

18.8.2.5 The component **localBSPHandle** shall contain the local BSP handle.

18.8.2.6 There shall not be two entries with the same value of the component **localBSPHandle**. There shall not be two entries with the same values of the components **hostingEndpointIRI** and **originalBSPHandle**.

18.8.3 Entry deletion

18.8.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **AttachSessionLocalReferences** table is deleted.

18.8.3.2 Let *hostingEndpointIRI* be the value of the component **hostingEndpointIRI** and *originalBSPHandle* the value of the component **originalBSPHandle** of the entry being deleted.

18.8.3.3 For each entry of the **GUIEventLocalSubscriptions** table (see 18.10) where the component **hostingEndpointIRI** has the value *hostingEndpointIRI* and the component **originalBSPHandle** is present and has the value *originalBSPHandle*, the framework shall perform the following actions (in order):

a) if *hostingEndpointIRI* is not the local endpoint IRI, then delete the entry of the **GUIEventLocalSubscriptions** table (subclause 18.10.3 applies), skipping the remaining actions;

b) create a temporary abstract value (say, *unsubscribeFromGUIEventsCallParams*) of type **UnsubscribeFromGUIEventsCallParams** (see 16.23.3), where:

1) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of the entry (presence and value);

2) if the optional component **originalBSPHandle** of the entry is absent, then the optional component **bspUuid** of *unsubscribeFromGUIEventsCallParams* shall be set from the component **bspProductUuid** of the entry; otherwise, it shall be absent;

3) the optional component **bspHandle** shall be set from the optional component **originalBSPHandle** of the entry (presence and value);

4) if the component **guiSelectEventHandlerAddress** of the entry has a value different from 0, then the component **guiSelectEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

5) if the component **guiStateEventHandlerAddress** of the entry has a value different from 0, then the component **guiStateEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

6) if the component **guiProgressEventHandlerAddress** of the entry has a value different from 0, then the component **guiProgressEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0; and

7) the components **guiSelectEventHandlerContext**, **guiStateEventHandlerContext**, and **guiProgressEventHandlerContext** shall be set to 0;

c) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnsubscribeFromGUIEvents** (see 16.23), where the parameters of the function call shall be set by converting from *unsubscribeFromGUIEventsCallParams* as specified in 16.23.6;

d) delete the entry of the **GUIEventLocalSubscriptions** table (subclause 18.10.3 applies).

18.8.3.4 If *hostingEndpointIRI* is the local endpoint IRI, then, for each entry of the **GUIEventRedirectors** table (see 18.12) where the component **originalBSPHandle** has the value *originalBSPHandle*, the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *unredirectCallParams*) of type **UnredirectGUIEvents-RequestParams** (see 16.29.2), where all the components shall be set from the components of the entry with the same names;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnredirectGUIEvents** (see 16.29), where the parameters of the function call shall be set by converting from *unredirectCallParams* as specified in 16.29.5;

c) delete the entry of the **GUIEventRedirectors** table (subclause 18.12.3 applies).

18.8.4 Life cycle

18.8.4.1 An entry can be added to the **AttachSessionLocalReferences** table:

– when a framework receives a call to the function **BioAPI_BSPAttach** from the local application (see 16.13).

18.8.4.2 An entry of the **AttachSessionLocalReferences** table can be deleted:

a) when a framework receives a call to the function **BioAPI_BSPDetach** from the local application (see 16.14); and

b) when an entry of the **RunningBSPLocalReferences** table is deleted (see 18.5.4.2).

18.9 The AttachSessionRemoteReferences conceptual table

This conceptual table shall be present in all slave endpoints and is defined in ASN.1 as follows:

```
AttachSessionRemoteReferences ::= SET OF
        reference AttachSessionRemoteReference

AttachSessionRemoteReference ::= SEQUENCE {
        referrerEndpointIRI     EndpointIRI,
        bspProductUuid          BioAPI-UUID,
        originalBSPHandle       BioAPI-HANDLE
    }
```

18.9.1 General

An entry of this table represents an attach session remote reference, that is a reference (established by a **bspLoad** request BIP message) held by a master endpoint to an attach session of a BSP running in the local endpoint.

18.9.2 Components

18.9.2.1 The component **referrerEndpointIRI** shall contain the endpoint IRI of the master endpoint that holds the attach session remote reference. This shall not be the local endpoint.

18.9.2.2 The component **bspProductUuid** shall contain the BSP product UUID of the registered BSP.

18.9.2.3 The component **originalBSPHandle** shall contain the attach session handle.

18.9.2.4 There shall not be two entries with the same value of the component **originalBSPHandle**.

18.9.3 Entry deletion

18.9.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **AttachSessionRemoteReferences** table is deleted.

18.9.3.2 Let *originalBSPHandle* be the value of the component **originalBSPHandle** of the entry being deleted.

18.9.3.3 For each entry of the **GUIEventRemoteSubscriptions** table (see 18.11) where the component **originalBSPHandle** is present and has the value *originalBSPHandle*, the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *unsubscribeFromGUIEventsCallParams*) of type **UnsubscribeFromGUIEventsCallParams** (see 16.23.3) where:

1) the optional component **guiEventSubscriptionUuid** shall be set from the optional component **guiEventSubscriptionUuid** of the entry (presence and value);

2) if the optional component **originalBSPHandle** of the entry is absent, then the optional component **bspUuid** of *unsubscribeFromGUIEventsCallParams* shall be set from the component **bspProductUuid** of the entry; otherwise, it shall be absent;

3) the optional component **bspHandle** shall be set from the optional component **originalBSPHandle** of the entry (presence and value);

4) if the component **guiSelectEventSubscribed** of the entry has the value **TRUE**, then the component **guiSelectEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

5) if the component **guiStateEventSubscribed** of the entry has the value **TRUE**, then the component **guiStateEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0;

6) if the component **guiProgressEventSubscribed** of the entry has the value **TRUE**, then the component **guiProgressEventHandlerAddress** of *unsubscribeFromGUIEventsCallParams* shall be set to an implementation-defined memory address different from 0, which shall be the same as that used in 16.22.5 b); otherwise, it shall be set to 0; and

7) the components **guiSelectEventHandlerContext**, **guiStateEventHandlerContext**, and **guiProgressEventHandlerContext** shall be set to 0;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnsubscribeFromGUIEvents** (see 16.23), where the parameters of the function call shall be set by converting from *unsubscribeFromGUIEventsCallParams* as specified in 16.23.6;

c) delete the entry of the **GUIEventRemoteSubscriptions** table (subclause 18.11.3 applies);

18.9.3.4 For each entry of the **GUIEventRedirectors** table (see 18.12) where the component **originalBSPHandle** has the value *originalBSPHandle*, the framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *unredirectCallParams*) of type **UnredirectGUIEvents-RequestParams** (see 16.29.2), where all the components shall be set from the components of the entry with the same names;

b) make an internal BioAPI function call (see 13.10) to the function **BioAPI_UnredirectGUIEvents** (see 16.29), where the parameters of the function call shall be set by converting from *unredirectCallParams* as specified in 16.29.5;

c) delete the entry of the **GUIEventRedirectors** table (subclause 18.12.3 applies).

18.9.4 Life cycle

18.9.4.1 An entry can be added to the **AttachSessionRemoteReferences** table:

– when a framework receives a **bspAttach** request BIP message from a master endpoint (see 16.13.2).

18.9.4.2 An entry of the **AttachSessionRemoteReferences** table can be deleted:

a) when a framework receives a **bspDetach** request BIP message from a master endpoint (see 16.14.2); and

b) when an entry of the **RunningBSPRemoteReferences** table is deleted (see 18.6.4.2).

18.10 The GUIEventLocalSubscriptions conceptual table

This conceptual table shall be present in all BIP endpoints and is defined in ASN.1 as follows:

```
GUIEventLocalSubscriptions ::= SET OF
        subscription GUIEventLocalSubscription

GUIEventLocalSubscription ::= SEQUENCE {
        guiEventSubscriptionUuid            BioAPI-UUID OPTIONAL,
        hostingEndpointIRI                  EndpointIRI,
        bspProductUuid                      BioAPI-UUID,
        useBSPAccessUuid                    BOOLEAN,
        originalBSPHandle                   BioAPI-HANDLE OPTIONAL,
        guiSelectEventHandlerAddress        MemoryAddress,
        guiSelectEventHandlerContext        MemoryAddress,
        guiStateEventHandlerAddress         MemoryAddress,
        guiStateEventHandlerContext         MemoryAddress,
        guiProgressEventHandlerAddress      MemoryAddress,
        guiProgressEventHandlerContext      MemoryAddress
    }
```

18.10.1 General

18.10.1.1 An entry of this table represents a GUI event local subscription, that is an obligation for the framework to make GUI event notification callbacks to the local application on certain incoming GUI event notifications (either a GUI event notification callback from a BSP or a GUI event notification BIP message from a slave endpoint) and incoming requests (either a GUI event notification request call from the local application or a GUI event notification request BIP message from a master endpoint).

18.10.1.2 An incoming call to **BioAPI_SubscribeToGUIEvents** can create a new GUI event local subscription entry even if the parameters of the call are the same as those of a previous incoming call to **BioAPI_SubscribeToGUIEvents**. An incoming call to **BioAPI_UnsubscribeFromGUIEvents** whose parameters match an existing GUI event local subscription entry can delete that entry. In case of multiple matches, any one of the matching entries can be deleted.

18.10.2 Components

18.10.2.1 The optional component **guiEventSubscriptionUuid**, if present, shall contain the GUI event subscription UUID. The absence of this component signifies a primary GUI event subscription.

18.10.2.2 The component **hostingEndpointIRI** shall contain the endpoint IRI of a BIP endpoint (either the local endpoint or a slave endpoint), and indicates that the subscription is limited to GUI events related to BSPs running in that BIP endpoint.

18.10.2.3 The component **bspProductUuid** shall contain a BSP product UUID, and indicates that the subscription is limited to GUI events related to the BSP with that BSP product UUID.

18.10.2.4 The component **useBSPAccessUuid** shall indicate whether the local application provided the BSP access UUID (rather than the BSP product UUID) to identify the BSP in the **BioAPI_SubscribeToGUIEvents** call (see 16.22) that caused the addition of the entry.

NOTE – A subsequent **BioAPI_UnsubscribeFromGUIEvents** call (see 16.23) has to provide the same UUID (either the BSP access UUID or the BSP product UUID) in order to match this GUI event local subscription. In addition, GUI event notification callbacks generated using this GUI event local subscription will pass the same UUID as input to the GUI event handlers of the local application.

18.10.2.5 The optional component **originalBSPHandle**, if present, shall contain an original BSP handle, and indicates that the subscription is limited to GUI event notifications carrying that BSP handle. The absence of this component indicates that the subscription is not limited to one particular attach session.

18.10.2.6 The component **guiSelectEventHandlerAddress** shall contain the callback address of a GUI select event handler of the local application. The value 0 indicates that GUI select events are not to be notified to the subscriber.

18.10.2.7 The component **guiSelectEventHandlerContext** shall contain the context address to be passed as input to the GUI select event handler.

18.10.2.8 The component **guiStateEventHandlerAddress** shall contain the callback address of a GUI state event handler of the local application. The value 0 indicates that GUI state events are not to be notified to the subscriber.

18.10.2.9 The component **guiStateEventHandlerContext** shall contain the context address to be passed as input to the GUI state event handler.

18.10.2.10 The component **guiProgressEventHandlerAddress** shall contain the callback address of a GUI progress event handler of the local application. The value 0 indicates that GUI progress events are not to be notified to the subscriber. At least one of the components **guiSelectEventHandlerAddress**, **guiStateEventHandlerAddress**, and **guiProgressEventHandlerAddress** shall have a value different from 0.

18.10.2.11 The component **guiProgressEventHandlerContext** shall contain the context address to be passed as input to the GUI progress event handler.

18.10.2.12 There may be multiple entries with the same values of one or more (or even all) of their components.

18.10.3 Entry deletion

18.10.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **GUIEventLocalSubscriptions** table is deleted.

18.10.3.2 No additional action is required.

18.10.4 Life cycle

18.10.4.1 An entry can be added to the **GUIEventLocalSubscriptions** table:

– when a framework receives a call to the function **BioAPI_SubscribeToGUIEvents** from the local application (see 16.22).

18.10.4.2 An entry of the **GUIEventLocalSubscriptions** table can be deleted:

a) when a framework receives a call to the function **BioAPI_UnsubscribeFromGUIEvents** from the local application (see 16.23);

b) when an entry of the **VisibleBSPRegistrations** table is deleted (see 18.3.4.2); and

c) when an entry of the **AttachSessionLocalReferences** table is deleted (see 18.8.4.2).

18.11 The GUIEventRemoteSubscriptions conceptual table

This conceptual table shall be present in all slave endpoints and is defined in ASN.1 as follows:

```
GUIEventRemoteSubscriptions ::= SET OF
        subscription GUIEventRemoteSubscription

GUIEventRemoteSubscription ::= SEQUENCE {
        subscriberEndpointIRI           EndpointIRI,
        guiEventSubscriptionUuid        BioAPI-UUID OPTIONAL,
        bspProductUuid                  BioAPI-UUID,
        originalBSPHandle               BioAPI-HANDLE OPTIONAL,
        guiSelectEventSubscribed        BOOLEAN,
        guiStateEventSubscribed         BOOLEAN,
        guiProgressEventSubscribed      BOOLEAN
    }
```

18.11.1 General

18.11.1.1 An entry of this table represents a GUI event remote subscription, that is an obligation for the framework to send GUI event notification BIP messages to a master endpoint on certain incoming GUI event notifications (a GUI event notification callback from a BSP) and incoming requests (either a GUI event notification request call from the local application or a GUI event notification request BIP message from a master endpoint).

18.11.1.2 An incoming **subscribeToGUIEvents** request BIP message can create a new GUI event remote subscription entry even if the parameter value of the BIP message is the same as that of a previous incoming **subscribeToGUIEvents** request BIP message. An incoming **unsubscribeFromGUIEvents** request BIP message whose components match an existing GUI event remote subscription entry can delete that entry. In case of multiple matches, any one of the matching entries can be deleted.

18.11.2 Components

18.11.2.1 The component **subscriberEndpointIRI** shall contain the endpoint IRI of the master endpoint that has subscribed to GUI events.

18.11.2.2 The optional component **guiEventSubscriptionUuid**, if present, shall contain the GUI event subscription UUID. The absence of this component signifies a primary GUI event subscription.

18.11.2.3 The component **bspProductUuid** shall contain a BSP product UUID, and indicates that the subscription is limited to GUI events related to the BSP with that BSP product UUID.

18.11.2.4 The optional component **originalBSPHandle**, if present, shall contain an original BSP handle, and indicates that the subscription is limited to GUI events related to the attach session identified by that BSP handle. The absence of this component indicates that the subscription is not limited to a particular attach session. This component shall only be present if the optional component **guiEventSubscriptionUuid** is absent.

18.11.2.5 The component **guiSelectEventSubscribed** indicates whether GUI select events are to be notified to the subscriber.

18.11.2.6 The component **guiStateEventSubscribed** indicates whether GUI state events are to be notified to the subscriber.

18.11.2.7 The component **guiProgressEventSubscribed** shall indicate whether GUI progress events are to be notified to the subscriber. At least one of the components **guiSelectEventSubscribed**, **guiStateEventSubscribed**, and **guiProgressEventSubscribed** shall have the value **TRUE**.

18.11.2.8 There may be multiple entries with the same values of one or more (or even all) of their components.

18.11.3 Entry deletion

18.11.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **GUIEventRemoteSubscriptions** table is deleted.

18.11.3.2 No additional action is required.

18.11.4 Life cycle

16.11.4.1 An entry can be added to the **GUIEventRemoteSubscriptions** table:

– when a framework receives a **subscribeToGUIEvents** request BIP message from a master endpoint (see 16.22.2).

18.11.4.2 An entry of the **GUIEventRemoteSubscriptions** table can be deleted:

a) when a framework receives an **unsubscribeFromGUIEvents** request BIP message from a master endpoint (see 16.23.2);

b) when an entry of the **MasterEndpoints** table is deleted (see 18.1.4.2);

c) when an entry of the **VisibleBSPRegistrations** table is deleted (see 18.3.4.2); and

d) when an entry of the **AttachSessionRemoteReferences** table is deleted (see 18.9.4.2).

18.12 The GUIEventRedirectors conceptual table

This conceptual table shall be present in all BIP endpoints and is defined in ASN.1 as follows:

```
GUIEventRedirectors ::= SET OF
        redirector GUIEventRedirector

GUIEventRedirector ::= SEQUENCE {
        referrerEndpointIRI             EndpointIRI,
        bspProductUuid                  BioAPI-UUID,
        originalBSPHandle               BioAPI-HANDLE,
        subscriberEndpointIRI           EndpointIRI,
        guiEventSubscriptionUuid        BioAPI-UUID,
        guiSelectEventRedirected        BOOLEAN,
        guiStateEventRedirected         BOOLEAN,
        guiProgressEventRedirected      BOOLEAN
    }
```

18.12.1 General

18.12.1.1 An entry of this table represents a GUI event redirector, that is an obligation for the framework to process GUI event notification callbacks received from a BSP running in the local endpoint (for a given attach session) as though they were incoming GUI event notification requests with certain parameters.

18.12.1.2 An incoming call to **BioAPI_RedirectGUIEvents** can create a new GUI event redirector entry even if the parameters of the call are the same as those of a previous call to **BioAPI_RedirectGUIEvents**. An incoming call to **BioAPI_UnredirectGUIEvents** whose components match an existing GUI event redirector entry can delete that entry. In case of multiple matches, any one of the matching entries can be deleted.

18.12.1.3 An incoming **redirectGUIEvents** request BIP message can create a new GUI event redirector entry even if the parameter value of the BIP message is the same as that of a previous incoming **redirectGUIEvents** request BIP message. An incoming **unredirectGUIEvents** request BIP message whose components match an existing GUI event redirector entry can delete that entry. In case of multiple matches, any one of the matching entries can be deleted.

18.12.2 Components

18.12.2.1 The component **referrerEndpointIRI** shall contain the endpoint IRI of the BIP endpoint that holds a reference (either an attach session local reference or an attach session remote reference) to the attach session in the GUI event redirector. This shall be either a master endpoint or the local endpoint.

18.12.2.2 The component **bspProductUuid** shall contain the BSP product UUID of the BSP in the GUI event redirector.

18.12.2.3 The component **originalBSPHandle** shall contain an original BSP handle, and indicates that the GUI event redirector is limited to GUI events related to the attach session identified by that BSP handle.

18.12.2.4 The component **subscriberEndpointIRI** shall contain the endpoint IRI of the BIP endpoint that is to receive the redirected GUI event notifications. This shall be either a master endpoint or the local endpoint.

18.12.2.5 The component **guiEventSubscriptionUuid** shall contain the GUI event subscription UUID to be assigned to the component **guiEventSubscriptionUuid** of the redirected GUI event notifications.

18.12.2.6 The component **guiSelectEventRedirected** shall indicate whether GUI select events are to be redirected to the subscriber.

18.12.2.7 The component **guiStateEventRedirected** shall indicate whether GUI state events are to be redirected to the subscriber.

18.12.2.8 The component **guiProgressEventRedirected** shall indicate whether GUI progress events are to be redirected to the subscriber. At least one of the components **guiSelectEventRedirected**, **guiStateEventRedirected**, and **guiProgressEventRedirected** shall have the value **TRUE**.

18.12.2.9 There may be multiple entries with the same values of one or more (or even all) of their components.

18.12.3 Entry deletion

18.12.3.1 This subclause only applies as explicitly invoked by other clauses of this Recommendation | International Standard when an entry of the **GUIEventRedirectors** table is deleted.

18.12.3.2 No additional action is required.

18.12.4 Life cycle

18.12.4.1 An entry can be added to the **GUIEventRedirectors** table:

a) when a framework receives a call to the function **BioAPI_RedirectGUIEvents** from the local application (see 16.28); and

b) when a framework receives a **redirectGUIEvents** request BIP message from a master endpoint (see 16.28.2).

18.12.4.2 An entry of the **GUIEventRedirectors** table can be deleted:

a) when a framework receives a call to the function **BioAPI_UnredirectGUIEvents** from the local application (see 16.29);

b) when a framework receives an **unredirectGUIEvents** request BIP message from a master endpoint (see 16.29.2);

c) when an entry of the **MasterEndpoints** table is deleted (see 18.1.4.2);

d) when an entry of the **AttachSessionLocalReferences** table is deleted (see 18.8.4.2); and

e) when an entry of the **AttachSessionRemoteReferences** table is deleted (see 18.9.4.2).

18.13 The ApplicationOwnedMemoryBlocks conceptual table

This conceptual table shall be present in all BIP endpoints and is defined in ASN.1 as follows:

```
ApplicationOwnedMemoryBlocks ::= SET OF
                memoryBlock ApplicationOwnedMemoryBlock

ApplicationOwnedMemoryBlock ::= SEQUENCE {
                address                 MemoryAddress
        }
```

18.13.1 General

18.13.1.1 An entry of this table represents a memory block allocated by the framework but owned by the local application.

18.13.1.2 This table keeps track of memory allocations done by the framework so that the memory blocks can be freed on either an incoming call to **BioAPI_Free** (see 16.58) or an incoming call to **BioAPI_Terminate** (see 16.3) from the local application.

18.13.2 Components

18.13.2.1 The component **address** shall contain the address of the memory block represented by the entry.

18.13.2.2 There shall not be two or more entries with the same value of the component **address**.

18.13.3 Life cycle

18.13.3.1 An entry can be added to the **ApplicationOwnedMemoryBlocks** table:

a) when an allocated variable or array is created while processing an incoming BioAPI function call from the local application (see 13.13).

18.13.3.2 An entry of the **ApplicationOwnedMemoryBlocks** table can be deleted:

a) when a framework receives a call to the function **BioAPI_Terminate** from the local application (see 16.3); and

b) when a framework receives a call to the function **BioAPI_Free** from the local application (see 16.58).

19 Converting between a C pointer variable and a corresponding ASN.1 component (1)

19.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard. It specifies conversion either:

a) between a C pointer variable that is a member of a larger structure and a component of the ASN.1 type corresponding to the C type of that structure; or

b) between a C pointer variable that is an input parameter of a function and a component of an ASN.1 type.

19.2 Call *Type* the type of the pointed-to variable as specified in the definition of the C pointer variable.

19.3 Conversion from the C pointer variable to the ASN.1 component shall be done as follows:

a) if the C pointer variable has the value **NULL** and the ASN.1 component is **OPTIONAL**, then the ASN.1 component shall be absent;

b) if the C pointer variable has the value **NULL** and the ASN.1 component is not **OPTIONAL**, then the C value is unconvertible and clause 33 applies;

c) if the C pointer variable has a value different from **NULL**, then the variable of type *Type* pointed to by the C pointer variable shall be converted to the ASN.1 component as specified in the subclause that is referenced in the invocation of this clause.

19.4 Conversion from the ASN.1 component to the C pointer variable shall be done as follows:

a) if the ASN.1 component is **OPTIONAL** and absent, then the C pointer variable shall be set to **NULL**;

b) if the ASN.1 component is present, then the C pointer variable shall be set to the address of a newly allocated variable of type *Type*, and the ASN.1 component shall be converted to that variable as specified in the subclause that is referenced in the invocation of this clause.

20 Converting between a C pointer variable and a corresponding ASN.1 component (2)

20.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard. It specifies conversion between a C pointer variable that is an output parameter of a function and a component of an ASN.1 type.

20.2 Call *Type* the type of the pointed-to variable as specified in the definition of the C pointer variable.

20.3 Conversion from the C pointer variable to the ASN.1 component shall be done as follows:

a) if the C pointer variable has the value **NULL**, then the ASN.1 component shall be absent;

NOTE – This case can only occur when the ASN.1 component is **OPTIONAL**.

b) if the C pointer variable has a value different from **NULL**, then the ASN.1 component shall be present and the variable of type *Type* pointed to by the C pointer variable shall be converted to that component as specified in the subclause that is referenced in the invocation of this clause.

20.4 Conversion from the ASN.1 component to the C pointer variable shall be done as follows:

a) if the C pointer variable has the value **NULL**, then no action shall be performed;

NOTE 1 – If the ASN.1 component is present, then it will be ignored (this situation never arises with a response or acknowledgement BIP message received from a conforming BIP endpoint).

b) if the ASN.1 component is **OPTIONAL** and absent, then no action shall be performed;

NOTE 2 – If the C pointer variable has a value different from **NULL**, then the variable of type *Type* pointed to by the C pointer variable will retain its current value (this situation never arises with a response or acknowledgement BIP message received from a conforming BIP endpoint).

c) if the C pointer variable has a value different from **NULL** and the ASN.1 component is present, then the ASN.1 component shall be converted to the variable of type *Type* pointed to by the C pointer variable as specified in the subclause that is referenced in the invocation of this clause.

21 Converting between a C pointer variable and a corresponding ASN.1 component (3)

21.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard. It specifies conversion between a C pointer variable that is an output parameter of a function and a component of an ASN.1 type.

NOTE – The corresponding ASN.1 component is always of type **BOOLEAN** and indicates whether the C pointer variable has the value **NULL**. All such boolean ASN.1 components have a name beginning with "**no-**", to convey the idea that the corresponding output parameter is not being requested.

21.2 Call *Type* the type of the pointed-to variable as specified in the definition of the C pointer variable.

21.3 Conversion from the C pointer variable to the ASN.1 component shall be done as follows: If the C pointer variable has the value **NULL**, then the ASN.1 component shall be set to **TRUE**. Otherwise, the ASN.1 component shall be set to **FALSE**.

NOTE – If the C pointer variable has a value different from **NULL**, then the value of the variable of type *Type* pointed to by the C pointer variable will be ignored.

21.4 Conversion from the ASN.1 component to the C pointer variable shall be done as follows: If the ASN.1 component has the value **TRUE**, then the C pointer variable shall be set to **NULL**. Otherwise, the C pointer variable shall be set to the address of a newly allocated variable of type *Type*, and the underlying memory block (as many bytes as the size of the variable) shall be filled with zeros.

22 Initializing and checking a C pointer variable having no corresponding ASN.1 component

22.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard. It specifies initialization and checking of a C pointer variable that is an output parameter of a function and has no corresponding component in a given ASN.1 type.

22.2 When converting from the parameters of a BioAPI function to an ASN.1 type, the following check shall be performed: If the C pointer variable has the value **NULL**, then the C value is unconvertible and clause 33 applies.

22.3 When converting from an ASN.1 type to the parameters of a BioAPI function, the C pointer variable shall be set to the address of a newly allocated variable of type *Type*, and the underlying memory block (as many bytes as the size of the variable) shall be filled with zeros.

NOTE – In some cases, the allocated variable is a pointer. In those cases, the underlying memory block will have the size of a pointer, and the pointer will be initialized to **NULL**.

23 Determining a hosting endpoint and a BSP product UUID from a BSP UUID

23.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard in order to determine the hosting endpoint of the BSP for an incoming call to a BioAPI function that has a parameter **BSPUuid** of type **const BioAPI_UUID***, as well as the BSP product UUID.

23.2 The framework shall perform the following actions (in order):

a) convert the parameter **BSPUuid** to a newly created temporary abstract value (say, *bspUuid*) of type **BioAPI-UUID** as specified in clause 19 in conjunction with 15.58;

b) search the **VisibleBSPRegistrations** table (see 18.3) for up to two entries where either the component **bspAccessUuid** or the component **bspProductUuid** of the entry has the value *bspUuid*;

c) if there is no matching entry, then conclude that the hosting endpoint cannot be determined;

NOTE 1 – The UUID provided by the local application is unknown, as it is neither a BSP product UUID nor a BSP access UUID.

d) if there is exactly one matching entry, then conclude that the hosting endpoint is the BIP endpoint identified by the component **hostingEndpointIRI** of that entry, and the BSP product UUID is the value of the component **bspProductUuid** of that entry;

e) if there are two or more matching entries, then conclude that the hosting endpoint cannot be determined.

NOTE 2 – This can only happen if the UUID provided by the local application is a BSP product UUID and the identified BSP is loadable or running in multiple hosting endpoints. The local application should have provided a BSP access UUID in the function call.

24 Determining a hosting endpoint and an original BSP handle from a local BSP handle

24.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard in order to determine the hosting endpoint of the BSP for an incoming call to a BioAPI function that has a parameter **BSPHandle** of type **BioAPI_HANDLE** (or **const BioAPI_HANDLE***), as well as the original BSP handle.

24.2 The framework shall perform the following actions (in order):

a) convert the parameter **BSPHandle** to a newly created temporary abstract value (say, *localBSPHandle*) of type **BioAPI-HANDLE** as specified in 15.42 (or as specified in clause 19 in conjunction with 15.42, if the type of the parameter is **const BioAPI_HANDLE***);

b) search the **AttachSessionLocalReferences** table (see 18.8) for an entry where the component **localBSPHandle** has the value *localBSPHandle*;

NOTE 1 – There can be at most one matching entry (see 16.13).

c) if there is no matching entry, then conclude that the hosting endpoint cannot be determined;

NOTE 2 – The BSP handle provided by the local application is unknown.

d) if there is one matching entry (say, *localReference*), then conclude that the hosting endpoint is that identified by the value of the component **hostingEndpointIRI** of *localReference*, and the original BSP handle is the value of the component **originalBSPHandle** of *localReference*.

25 Converting BSP UUIDs

25.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard in order to convert between a C function parameter of type **const BioAPI_UUID*** (containing either a BSP product UUID or a BSP access UUID) and an ASN.1 component of type **BioAPI-UUID** (containing a BSP product UUID).

25.2 Conversion from the C parameter to the ASN.1 component shall be done by setting the ASN.1 abstract value to the BSP product UUID determined as specified in clause 23.

25.3 Conversion from the ASN.1 component to the C parameter shall be done as specified in 15.58.

26 Converting BSP handles

26.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard in order to convert between a C function parameter of type **BioAPI_HANDLE** (containing a local BSP handle) and an ASN.1 component of type **BioAPI-HANDLE** (containing an original BSP handle).

26.2 Conversion from the C parameter to the ASN.1 component shall be done by setting the ASN.1 abstract value to the original BSP handle determined as specified in clause 24.

26.3 Conversion from the ASN.1 component to the C parameter shall be done as specified in 15.42.

27 Processing an incoming function call by exchanging a request/response BIP message pair with a slave endpoint

27.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard in order to process an incoming function call by sending a request BIP message to a slave endpoint and receiving the corresponding response BIP message from it.

27.2 The framework shall perform the following actions (in order):

a) create a temporary abstract value (say, *outgoingRequestParams*) of a request BIP message parameter ASN.1 type (which is specified in the invocation of this clause) by converting from the parameters of the function call using the subclause specified in the invocation of this clause;

b) create and send a request BIP message (see 13.2) of the BIP message type specified in the invocation of this clause, with the slave endpoint IRI set to the endpoint IRI of the slave endpoint and the parameter value set to *outgoingRequestParams*;

c) receive a corresponding response BIP message (see 13.6);

d) if the return value of the response BIP message is not 0, then return that value to the local application, and skip the remaining actions;

e) set the output parameters of the function call by converting from the parameter value of the response BIP message using the subclause specified in the invocation of this clause;

f) return the value 0 to the local application.

28 Processing an incoming request BIP message via an internal BioAPI function call

28.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard in order to process an incoming request BIP message from a given master endpoint by making an internal BioAPI function call (see 13.10), and then send a corresponding response BIP message.

28.2 The framework shall perform the following actions (in order):

a) make an internal BioAPI function call (see 13.10) to the BioAPI function specified in the invocation of this clause, where the parameters of the function call shall be set by converting from a request BIP message parameter ASN.1 type using the subclause specified in the invocation of this clause;

b) if the return value of the internal call is not 0, then create and send a corresponding response BIP message (see 13.3) with the return value set to that value, skipping the remaining actions;

c) create a temporary abstract value (say, *incomingResponseParams*) of a response BIP message parameter ASN.1 type (specified in the invocation of this clause) by converting from the output parameters of the internal call using the subclause specified in the invocation of this clause;

d) create and send a corresponding response BIP message (see 13.3) with the parameter value set to *incomingResponseParams* and the return value set to 0.

29 Notifying a unit event to zero or more subscribers

29.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard in order to notify a unit event based on a value (say, *eventInfo*) of type **UnitEventInfo** to zero or more subscribers, which may be:

a) zero or more unit event handlers of the local application; or

b) zero or more master endpoints,

or both.

29.2 The framework shall perform the following actions (in order):

a) search the **UnitEventNotificationDisablers** table (see 18.7) for an entry where:

1) the component **referrerEndpointIRI** is set to the local endpoint IRI;

2) the component **bspProductUuid** has the same value as the component **bspProductUuid** of *eventInfo*; and

3) the component **unitEventTypes** has a value indicating that the type of event notification of *eventInfo* is disabled;

b) if there is a matching entry, then skip the remaining actions (continue with the next subclause);

c) search the **RunningBSPLocalReferences** table (see 18.5) for all entries where:

1) the components **hostingEndpointIRI** and **bspProductUuid** have the same values as the components of *eventInfo* with the same names; and

2) the component **unitEventHandlerAddress** has a value different from 0;

d) for each matching entry (say, *localSubscription*), in any order do the following: Create a temporary abstract value (say, *outgoingCallbackParams*) of type **UnitEventHandlerCallbackParams** (see 17.1.4) where:

1) the components **unitEventHandlerAddress** and **unitEventHandlerContext** shall be set from the components of *localSubscription* with the same names;

2) if the component **useBSPAccessUuid** of *localSubscription* has the value **FALSE**, then the component **bspUuid** of *outgoingCallbackParams* shall be set from the component **bspProductUuid** of *eventInfo*; otherwise, it shall be set from the component **bspAccessUuid** of the entry of the **VisibleBSPRegistrations** table (see 18.3) where the components **bspProductUuid** and **hostingEndpointIRI** of the entry have the same values as the components of *eventInfo* with the same names; and

3) the remaining components shall be set from the components of *eventInfo* with the same names;

and make a call to a callback function **BioAPI_EVENT_HANDLER** of the local application, where the callback address and the parameters of the function call shall be set by converting from *outgoingCallbackParams* as specified in 17.1.8. Ignore the value returned by each of these calls, but wait for each call to return before making the next call.

29.3 If the component **hostingEndpointIRI** of *eventInfo* contains the local endpoint IRI, then, for each entry (say, *masterEndpoint*) of the **MasterEndpoints** table (see 18.1), the framework shall perform the following actions (in order):

a) search the **UnitEventNotificationDisablers** table (see 18.7) for an entry where:

1) the component **referrerEndpointIRI** has the same value as the component **masterEndpointIRI** of *masterEndpoint*;

2) the component **bspProductUuid** has the same value as the component **bspProductUuid** of *eventInfo*; and

3) the component **unitEventTypes** has a value indicating that the type of event notification of *eventInfo* is disabled;

b) if there is a matching entry, then skip the remaining actions (continue with the next entry of the **MasterEndpoints** table);

c) search the **RunningBSPRemoteReferences** table (see 18.6) for an entry where:

1) the component **referrerEndpointIRI** has the same value as the component **masterEndpointIRI** of *masterEndpoint*;

2) the component **bspProductUuid** has the same value as the component **bspProductUuid** of *eventInfo*; and

3) the component **unitEventSubscription** has the value **TRUE**;

d) if there is no matching entry, then skip the remaining actions (continue with the next entry of the **MasterEndpoints** table);

e) create a temporary abstract value (say, *outgoingNotificationParams*) of type **UnitEvent-NotificationParams** (see 17.1.3), where all the components shall be set from the components of *eventInfo* with the same names;

f) create and send a **unitEvent** notification BIP message (see 13.4 and 17.1.3) with the master endpoint IRI set from the component **masterEndpointIRI** of *masterEndpoint* and the parameter value set to *outgoingNotificationParams*.

NOTE – At most one notification BIP message is sent to each master endpoint even if there are multiple matching entries in the **RunningBSPRemoteReferences** table for the same master endpoint.

30 Notifying a GUI select event to a subscriber

30.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard in order to notify a GUI select event based on a value (say, *eventInfo*) of type **GUISelectEventInfo** to a subscriber, which may be either:

a) a GUI select event handler of the local application; or

b) a master endpoint,

and to determine the acknowledgement parameter value (*incomingAcknowledgmentParams*) and the acknowledgement return value resulting from the notification.

30.2 If, by applying the following subclauses, the framework determines that there is no subscriber, then it shall create:

a) a temporary abstract value *incomingAcknowledgementParams* of type **GUISelectEvent-AcknowledgementParams** (see 17.2.3) where the component **selectedInstances** is set from the component **selectableInstances** of *eventInfo* and the component response is set to default; and

b) a temporary abstract value *incomingReturnValue* of type **BioAPI-RETURN** (see 15.52) set to 0.

30.3 If the component **subscriberEndpointIRI** of *eventInfo* contains the local endpoint IRI, then the framework shall perform the following actions (in order):

a) search the **GUIEventLocalSubscriptions** table (see 18.10) for an entry (say, *localSubscription*) where:

1) the components **hostingEndpointIRI** and **bspProductUuid** have the same values as the components of *eventInfo* with the same names;

2) the optional component **guiEventSubscriptionUuid** has the same presence and value as the optional component **guiEventSubscriptionUuid** of *eventInfo*;

3) if the optional component **originalBSPHandle** is present, then the optional component **originalBSPHandle** of *eventInfo* is also present and the two components have the same value; and

4) the component **guiSelectEventHandlerAddress** has a value different from 0;

b) if there is no matching entry, then conclude that there is no subscriber (see 30.2) and skip the remaining actions;

NOTE – Here is at most one matching entry.

c) create a temporary abstract value (say, *outgoingCallbackParams*) of type **GUISelectEventHandlerCallbackParams** (see 17.2.4) where:

1) the components **guiSelectEventHandlerAddress** and **guiSelectEventHandlerContext** shall be set from the components of *localSubscription* with the same names;

2) if the component **useBSPAccessUuid** of *localSubscription* has the value **FALSE**, then the component **bspUuid** of *outgoingCallbackParams* shall be set from the component **bspProductUuid** of *eventInfo*; otherwise, it shall be set from the component **bspAccessUuid** of the entry of the **VisibleBSPRegistrations** table (see 18.3) where the components **bspProductUuid** and **hostingEndpointIRI** of the entry have the same values as the components of *eventInfo* with the same names;

3) if the optional component **originalBSPHandle** of *eventInfo* is absent, then the optional component **bspHandle** of *outgoingCallbackParams* shall also be absent; otherwise, it shall be set from the component **localBSPHandle** of the entry of the **AttachSessionLocalReferences** table (see 18.8) where the components **originalBSPHandle** and **hostingEndpointIRI** of the entry have the same values as the components of *eventInfo* with the same names; and

4) the remaining components shall be set from the components of *eventInfo* with the same names;

d) make a call to a callback function **BioAPI_GUI_SELECT_EVENT_HANDLER** of the local application, where the callback address and the parameters of the function call shall be set by converting from *outgoingCallbackParams* as specified in 17.2.10;

e) create a temporary abstract value (say, *incomingAcknowledgementParams*) of type **GUISelectEvent-AcknowledgementParams** (see 17.2.3) by converting from the output parameters of the function call as specified in 17.2.12;

f) create a temporary abstract value (say, *incomingReturnValue*) of type **BioAPI-RETURN** (see 15.52) by converting from the return value of the function call as specified in 15.1.5.

30.4 If the component **subscriberEndpointIRI** of *eventInfo* contains a master endpoint IRI, then the framework shall perform the following actions (in order):

a) search the **GUIEventRemoteSubscriptions** table (see 18.11) for an entry where:

1) the components **subscriberEndpointIRI** and **bspProductUuid** have the same values as the components of *eventInfo* with the same names;

2) the optional component **guiEventSubscriptionUuid** has the same presence and value as the optional component **guiEventSubscriptionUuid** of *eventInfo*;

3) the optional component **originalBSPHandle** has the same presence and value as the optional component **originalBSPHandle** of *eventInfo*; and

4) the component **guiSelectEventSubscribed** has the value **TRUE**;

b) if there is no such entry, then conclude that there is no subscriber (see 30.2) and skip the remaining actions;

c) create a temporary abstract value (say, *outgoingNotificationParams*) of type **GUISelectEvent-NotificationParams** (see 17.2.3), where all the components shall be set from the components of *eventInfo* with the same names;

d) create and send a **guiSelectEvent** notification BIP message (see 13.4 and 17.2.3) with the master endpoint IRI set from the component **subscriberEndpointIRI** of *eventInfo* and the parameter value set to *outgoingNotificationParams*;

e) receive a corresponding **guiSelectEvent** acknowledgement BIP message (see 13.7);

f) let *incomingAcknowledgementParams* be the parameter value (of type **GUISelectEvent-AcknowledgementParams** – see 17.2.3) of the **guiSelectEvent** acknowledgement BIP message;

g) let *incomingReturnValue* be the return value (of type **BioAPI-RETURN** – see 15.52) of the acknowledgement BIP message.

31 Notifying a GUI state event to a subscriber

31.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard in order to notify a GUI state event based on a value (say, *eventInfo*) of type **GUIStateEventInfo** to a subscriber, which may be either:

a) a GUI state event handler of the local application; or

b) a master endpoint,

and to determine the acknowledgement parameter value (*incomingAcknowledgementParams*) and the acknowledgement return value resulting from the notification.

31.2 If, by applying the following subclauses, the framework determines that there is no subscriber, then it shall create:

a) a temporary abstract value *incomingAcknowledgementParams* of type **GUIStateEvent-AcknowledgementParams** (see 17.3.3) where the component **enrollSampleIndexToRecapture** is set to 0 and the component response is set to default; and

b) a temporary abstract value *incomingReturnValue* of type **BioAPI-RETURN** (see 15.52) set to 0.

31.3 If the component **subscriberEndpointIRI** of *eventInfo* contains the local endpoint IRI, then the framework shall perform the following actions (in order):

a) search the **GUIEventLocalSubscriptions** table (see 18.10) for an entry (say, *localSubscription*) where:

1) the components **hostingEndpointIRI** and **bspProductUuid** have the same values as the components of *eventInfo* with the same names;

2) the optional component **guiEventSubscriptionUuid** has the same presence and value as the optional component **guiEventSubscriptionUuid** of *eventInfo*;

3) if the optional component **originalBSPHandle** is present, then the optional component **originalBSPHandle** of *eventInfo* is also present and the two components have the same value; and

4) the component **guiStateEventHandlerAddress** has a value different from 0;

b) if there is no matching entry, then conclude that there is no subscriber (see 30.2) and skip the remaining actions;

NOTE – There is at most one matching entry.

c) create a temporary abstract value (say, *outgoingCallbackParams*) of type **GUIStateEventHandlerCallbackParams** (see 17.3.4) where:

1) the components **guiStateEventHandlerAddress** and **guiStateEventHandlerContext** shall be set from the components of *localSubscription* with the same names;

2) if the component **useBSPAccessUuid** of *localSubscription* has the value **FALSE**, then the component **bspUuid** of *outgoingCallbackParams* shall be set from the component **bspProductUuid** of *eventInfo*; otherwise, it shall be set from the component **bspAccessUuid** of the entry of the **VisibleBSPRegistrations** table (see 18.3) where the components **bspProductUuid** and **hostingEndpointIRI** of the entry have the same values as the components of *eventInfo* with the same names;

3) if the optional component **originalBSPHandle** of *eventInfo* is absent, then the optional component **bspHandle** of *outgoingCallbackParams* shall also be absent; otherwise, it shall be set from the component **localBSPHandle** of the entry of the **AttachSessionLocalReferences** table (see 18.8) where the components **originalBSPHandle** and **hostingEndpointIRI** of the entry have the same values as the components of *eventInfo* with the same names; and

4) the remaining components shall be set from the components of *eventInfo* with the same names;

d) make a call to a callback function **BioAPI_GUI_STATE_EVENT_HANDLER** of the local application, where the callback address and the parameters of the function call shall be set by converting from *outgoingCallbackParams* as specified in 17.3.10;

e) create a temporary abstract value (say, *incomingAcknowledgementParams*) of type **GUIStateEvent-AcknowledgementParams** (see 17.3.3) by converting from the output parameters of the function call as specified in 17.3.12;

f) create a temporary abstract value (say, *incomingReturnValue*) of type **BioAPI-RETURN** (see 15.52) by converting from the return value of the function call as specified in 15.1.5.

31.4 If the component **subscriberEndpointIRI** of *eventInfo* contains a master endpoint IRI, then the framework shall perform the following actions (in order):

a) search the **GUIEventRemoteSubscriptions** table (see 18.11) for an entry where:

1) the components **subscriberEndpointIRI** and **bspProductUuid** have the same values as the components of *eventInfo* with the same names;

2) the optional component **guiEventSubscriptionUuid** has the same presence and value as the optional component **guiEventSubscriptionUuid** of *eventInfo*;

3) the optional component **originalBSPHandle** has the same presence and value as the optional component **originalBSPHandle** of *eventInfo*; and

4) the component **guiStateEventSubscribed** has the value **TRUE**;

b) if there is no such entry, then conclude that there is no subscriber (see 30.2) and skip the remaining actions;

c) create a temporary abstract value (say, *outgoingNotificationParams*) of type **GUIStateEvent-NotificationParams** (see 17.3.3), where all the components shall be set from the components of *eventInfo* with the same names;

d) create and send a **guiStateEvent** notification BIP message (see 13.4 and 17.3.3) with the master endpoint IRI set from the component **subscriberEndpointIRI** of *eventInfo* and the parameter value set to *outgoingNotificationParams*;

e) receive a corresponding **guiStateEvent** acknowledgement BIP message (see 13.7);

f) let *incomingAcknowledgementParams* be the parameter value (of type **GUIStateEvent-AcknowledgementParams** – see 17.3.3) of the **guiStateEvent** acknowledgement BIP message;

g) let *incomingReturnValue* be the return value (of type **BioAPI-RETURN** – see 15.52) of the acknowledgement BIP message.

32 Notifying a GUI progress event to a subscriber

32.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard in order to notify a GUI progress event based on a value (say, *eventInfo*) of type **GUIProgressEventInfo** to a subscriber, which may be either:

a) a GUI progress event handler of the local application; or

b) a master endpoint,

and to determine the acknowledgement parameter value (*incomingAcknowledgementParams*) and the acknowledgement return value resulting from the notification.

32.2 If, by applying the following subclauses, the framework determines that there is no subscriber, then it shall create:

a) a temporary abstract value *incomingAcknowledgementParams* of type **GUIProgressEvent-AcknowledgementParams** (see 17.4.3) where the component response is set to default; and

b) a temporary abstract value *incomingReturnValue* of type **BioAPI-RETURN** (see 15.52) set to 0.

32.3 If the component **subscriberEndpointIRI** of *eventInfo* contains the local endpoint IRI, then the framework shall perform the following actions (in order):

a) search the **GUIEventLocalSubscriptions** table (see 18.10) for an entry (say, *localSubscription*) where:

1) the components **hostingEndpointIRI** and **bspProductUuid** have the same values as the components of *eventInfo* with the same names;

2) the optional component **guiEventSubscriptionUuid** has the same presence and value as the optional component **guiEventSubscriptionUuid** of *eventInfo*;

3) if the optional component **originalBSPHandle** is present, then the optional component **originalBSPHandle** of *eventInfo* is also present and the two components have the same value; and

4) the component **guiProgressEventHandlerAddress** has a value different from 0;

b) if there is no matching entry, then conclude that there is no subscriber (see 30.2) and skip the remaining actions;

NOTE – There is at most one matching entry.

c) create a temporary abstract value (say, *outgoingCallbackParams*) of type **GUIProgressEventHandlerCallbackParams** (see 17.4.4) where:

1) the components **guiProgressEventHandlerAddress** and **guiProgressEventHandlerContext** shall be set from the components of *localSubscription* with the same names;

2) if the component **useBSPAccessUuid** of *localSubscription* has the value **FALSE**, then the component **bspUuid** of *outgoingCallbackParams* shall be set from the component **bspProductUuid** of *eventInfo*; otherwise, it shall be set from the component **bspAccessUuid** of the entry of the **VisibleBSPRegistrations** table (see 18.3) where the components **bspProductUuid** and **hostingEndpointIRI** of the entry have the same values as the components of *eventInfo* with the same names;

3) if the optional component **originalBSPHandle** of *eventInfo* is absent, then the optional component **bspHandle** of *outgoingCallbackParams* shall also be absent; otherwise, it shall be set from the component **localBSPHandle** of the entry of the **AttachSessionLocalReferences** table (see 18.8) where the components **originalBSPHandle** and **hostingEndpointIRI** of the entry have the same values as the components of *eventInfo* with the same names; and

4) the remaining components shall be set from the components of *eventInfo* with the same names;

d) make a call to a callback function **BioAPI_GUI_PROGRESS_EVENT_HANDLER** of the local application, where the callback address and the parameters of the function call shall be set by converting from *outgoingCallbackParams* as specified in 17.4.10;

e) create a temporary abstract value (say, *incomingAcknowledgementParams*) of type **GUIProgressEvent-AcknowledgementParams** (see 17.4.3) by converting from the output parameters of the function call as specified in 17.4.12;

f) create a temporary abstract value (say, *incomingReturnValue*) of type **BioAPI-RETURN** (see 15.52) by converting from the return value of the function call as specified in 15.1.5.

32.4 If the component **subscriberEndpointIRI** of *eventInfo* contains a master endpoint IRI, then the framework shall perform the following actions (in order):

a) search the **GUIEventRemoteSubscriptions** table (see 18.11) for an entry where:

1) the components **subscriberEndpointIRI** and **bspProductUuid** have the same values as the components of *eventInfo* with the same names;

2) the optional component **guiEventSubscriptionUuid** has the same presence and value as the optional component **guiEventSubscriptionUuid** of *eventInfo*;

3) the optional component **originalBSPHandle** has the same presence and value as the optional component **originalBSPHandle** of *eventInfo*; and

4) the component **guiProgressEventSubscribed** has the value **TRUE**;

b) if there is no such entry, then conclude that there is no subscriber (see 30.2) and skip the remaining actions;

c) create a temporary abstract value (say, *outgoingNotificationParams*) of type **GUIProgressEvent-NotificationParams** (see 17.4.3), where all the components shall be set from the components of *eventInfo* with the same names;

d) create and send a **guiProgressEvent** notification BIP message (see 13.4 and 17.4.3) with the master endpoint IRI set from the component **subscriberEndpointIRI** of *eventInfo* and the parameter value set to *outgoingNotificationParams*;

e) receive a corresponding **guiProgressEvent** acknowledgement BIP message (see 13.7);

f) let *incomingAcknowledgementParams* be the parameter value (of type **GUIProgressEvent-AcknowledgementParams** – see 17.4.3) of the **guiProgressEvent** acknowledgement BIP message;

g) let *incomingReturnValue* be the return value (of type **BioAPI-RETURN** – see 15.52) of the acknowledgement BIP message.

33 Handling unconvertible C values

33.1 This clause only applies as explicitly invoked by other clauses of this Recommendation | International Standard in order to handle an unconvertible C value encountered during a conversion from a C type to an ASN.1 type.

33.2 When an unconvertible C value is detected during the processing of a request BIP message received from a master endpoint, the framework shall create and send a corresponding response BIP message (see 13.3) with the return value set to **BioAPIERR_UNCONVERTIBLE_VALUE**, skipping the remaining processing.

33.3 When an unconvertible C value is detected during the processing of an incoming call from the local application, the framework shall return the value **BioAPIERR_UNCONVERTIBLE_VALUE** to the local application, skipping the remaining processing.

33.4 When an unconvertible C value is detected during the processing of a notification BIP message received from a slave endpoint and having an associated acknowledgement BIP message type, the framework shall create and send a corresponding acknowledgement BIP message (see 13.5) with the return value set to **BioAPIERR_UNCONVERTIBLE_VALUE**, skipping the remaining processing.

33.5 When an unconvertible C value is detected during the processing of an incoming callback from a BSP, the framework shall return the value **BioAPIERR_UNCONVERTIBLE_VALUE** to the BSP, skipping the remaining processing.

Annex A

Specification of the TCP/IP binding

(This annex forms an integral part of this Recommendation | International Standard)

A.1 General

The BIP protocol specification covers levels 5 to 7 (session layer, presentation layer, application layer) of the ISO/IEC OSI-7-layer model to enable its use in all networks that support layers up to the transport layer. The binding of the BIP protocol to the transport layer interface provided by TCP/IP is specified in this annex.

A.2 Transport-level message

A transport-level message for the binding specified in this annex is defined in ASN.1 as follows:

```
BIP-TCPIP {joint-iso-itu-t bip(41) modules(0) bip-tcpip(1) version1(1)}
DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
IMPORTS BIPMessage FROM BIP{joint-iso-itu-t bip(41) modules(0) bip(0) version1(1)};

TCPIPBIPMessage ::=   SEQUENCE {
        magicNumber             OCTET STRING(SIZE(4))('3AC49E70'H),
        version                 INTEGER{version-1(1)}(0..255),
        content                 CHOICE {
                bIPMessage      OCTET STRING(CONTAINING BIPMessage
                                ENCODED BY basic-per-aligned),
                keepalive       NULL,
                requestLinkChannelOnSpecifiedPort   INTEGER(0..65535),
                requestLinkChannel   NULL
                }
        }

basic-per-aligned    OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1) packed-encoding(3) basic(0) aligned(0)}

END
```

The alternatives of the component **content** are defined in Table A.1.

Table A.1 – Transport-level message types in the TCP/IP binding of BIP

Alternative	Allowed sender of the message	Purpose of the message	Content of the field content
bipMessage	Either endpoint of a link channel that is being supported by the TCP/IP connection	Carries an encoded BIP message	BIP message (ASN.1 abstract value) encoded in Aligned PER (see ITU-T Rec. X.691 | ISO/IEC 8825-2)
Keepalive	Either endpoint of a link channel that is being supported by the TCP/IP connection	Informs the remote endpoint that the sending endpoint is still present and active	Empty
requestLinkChannel OnSpecifiedPort	The master endpoint in a request/response link channel that is being supported by this TCP/IP connection (only when a notification acknowledgement link channel has not yet been established as part of the same BIP link)	Requests the slave endpoint to establish a notification/acknowledgement link channel by opening a separate TCP/IP connection to the specified port of the master endpoint	2-byte unsigned integer containing a port number
requestLinkChannel	The master endpoint in a request/response link channel that is being supported by this TCP/IP connection (only when a notification/acknowledgement link channel has not yet been established as part of the same BIP link)	Requests the slave endpoint to establish a notification/acknowledgement link channel by utilizing the same TCP/IP which will then be supporting both link channels	Empty

A.3 TCP/IP connection between two BIP endpoints

A.3.1 A TCP/IP connection between two BIP endpoints may support any of the following:

a) a request/response link channel;

b) a notification/acknowledgement link channel; or

c) a request/response link channel and a notification/acknowledgement link channel that are part of the same BIP link.

A.3.2 The recommended port number for a TCP/IP connection supporting a request/response link channel is 4376. The recommended port number for a TCP/IP connection supporting a notification/acknowledgement link channel is 4376.

A.3.3 A transport-level message containing the alternative keepalive (see Table A.1) may be sent by either end of the TCP/IP connection. The frequency of such messages is implementation-defined.

A.4 Role of endpoint

A.4.1 If a framework wishes to make itself available to play the role of slave in a potential request/response link channel over a TCP/IP connection, it shall start listening to incoming connection requests on the recommended TCP/IP port (see A.3) or a port number determined by other means.

A.4.2 If a framework wishes to play the role of master in a request/response link channel over a TCP/IP connection with a given BIP endpoint, it shall attempt to open a TCP/IP connection with that BIP endpoint using either the recommended port number (see A.3) or a port number determined by other means.

A.4.3 If a framework is playing the role of master in a request/response link channel with a given BIP endpoint (over any transport protocol) and a notification/acknowledgement link channel has not yet been established in the same BIP link, the framework may do any one of the following:

a) make itself available to establish a notification/acknowledgement link channel with the slave endpoint using a different transport protocol; or

b) (if the request/response link channel uses a different transport protocol) start listening to incoming connection requests from the slave endpoint on the recommended TCP/IP port (see A.3); or a port determined by other means;

c) (if the request/response link channel is over TCP/IP) start listening to incoming connection requests from the slave endpoint on an arbitrary TCP/IP port at the same IP address, and send a transport-level message containing the alternative **requestLinkChannelOnSpecifiedPort** (see Table A.1) to the slave endpoint specifying the port number; or

d) (if the request/response link channel is over TCP/IP) send a transport-level message containing the alternative **requestLinkChannel** (see Table A.1) to the slave endpoint; or

e) do nothing (there will be no notification/acknowledgement link channel in the BIP link).

A.4.4 If a framework is playing the role of slave in a request/response link channel with a given BIP endpoint (over any transport protocol) and a notification/acknowledgement link channel has not yet been established in the same BIP link, the framework may do any one of the following:

a) attempt to establish a notification/acknowledgement link channel with the master endpoint using a different transport protocol; or

b) (if the request/response link channel uses a different transport protocol) attempt to open a TCP/IP connection with the master endpoint using either the recommended port number (see A.3) or a port number determined by other means;

c) (if the request/response link channel is over TCP/IP and a transport-level message containing the alternative **requestLinkChannelOnSpecifiedPort** (see Table A.1) has been received from the master endpoint through this TCP/IP connection) attempt to open a new TCP/IP connection with the master endpoint using the port number specified in the transport-level message containing the alternative **requestLinkChannelOnSpecifiedPort** (see Table A.1) received from the master endpoint;

d) (if the request/response link channel is over TCP/IP and a transport-level message containing the alternative **requestLinkChannel** (see Table A.1) has been received from the master endpoint through this TCP/IP connection) note that this TCP/IP connection will now support the notification/acknowledgement link channel as well as the request/response link channel;

e) do nothing (there will be no notification/acknowledgement link channel in the BIP link).

A.4.5 A framework playing the role of master shall send at most one transport-level message containing alternative **requestLinkChannelOnSpecifiedPort** or **requestLinkChannel** (see Table A.1) through a TCP/IP connection.

A.4.6 If a framework playing the role of slave has already received a transport-level message containing alternative **requestLinkChannelOnSpecifiedPort** or **requestLinkChannel** (see Table A.1) through a TCP/IP connection, it shall ignore any subsequent messages of either type received through the same TCP/IP connection.

A.4.7 A framework playing the role of master in a request/response link channel over a TCP/IP connection shall not close that TCP/IP connection before receiving from the slave endpoint either:

a) a **masterDeletionEvent** notification BIP message;

b) a **deleteMaster** response BIP message through the request/response link channel; or

c) a **masterDeletionEvent** notification BIP message through the notification/acknowledgement link channel that is part of the same BIP link (regardless of the transport protocol used for that link channel).

A.4.8 A framework playing the role of slave in a request/response link channel over a TCP/IP connection shall not close that TCP/IP connection before sending to the master endpoint a BIP message as specified in A.4.7 (a) and (b).

A.4.9 A framework playing the role of master in a notification/acknowledgement link channel over a TCP/IP connection shall not close that TCP/IP connection before receiving from the slave endpoint either:

a) a **deleteMaster** response BIP message through the request/response link channel that is part of the same BIP link (regardless of the transport protocol used for that link channel); or

b) a **masterDeletionEvent** notification BIP message through the notification/acknowledgement link channel.

A.4.10 A framework playing the role of slave in a notification/acknowledgement link channel over a TCP/IP connection shall not close that TCP/IP connection before sending to the master endpoint a BIP message as specified in A.4.9 (a) and (b).

A.4.11 When a framework playing the role of slave in a request/response link channel over a TCP/IP connection detects a failure or a premature closure of the TCP/IP connection, it shall behave as though it has received a **deleteMaster** request BIP message from the master endpoint.

A.4.12 When a framework playing the role of master in a notification/acknowledgement link channel over a TCP/IP connection detects a failure or a premature closure of the TCP/IP connection, it shall behave as though it has received a **masterDeletionEvent** notification BIP message from the slave endpoint.

A.5 Closing the connection on errors

A.5.1 If a framework receives a transport-level message containing the alternative **bipMessage** (containing an encoded BIP message) and the decoding of the BIP message causes decoding errors or the encoded BIP message appears to be truncated or shorter than the length in the field content length, then the BIP endpoint shall close the TCP/IP connection.

A.5.2 If a framework receives a transport-level message that is malformed, it shall close the TCP/IP connection.

A.6 Transport of BIP messages

Every BIP message sent through a link channel over TCP/IP shall be carried within a transport-level message containing the alternative **bipMessage** as specified in Table A.1 and shall be encoded in aligned PER (see ITU-T Rec. X.691 | ISO/IEC 8825-2).

A.7 Usage of IRIs

Endpoint IRIs may have any form and are not constrained by the TCP/IP binding (but see A.4.4). A general mapping between IRIs and IP addresses is not specified in this Recommendation | International Standard.

Annex B

Specification of discovery and announcement in TCP/IP binding

(This annex forms an integral part of this Recommendation | International Standard)

B.1 General

This annex contains additional provisions for the TCP/IP binding specified in Annex A.

B.2 The PnP mechanisms

The PnP mechanisms in this annex are applicable only when the BIP system is connected to an Ethernet-/IP-based network satisfying the following requirements:

Physical and Data Link layer (layers 1 and 2)

The Ethernet Adapter shall fulfil the following requirements:

a) 10 Mbit shall be supported;

b) Support for 100 Mbit and/or 1 Gbit operation is optional;

c) Auto-negotiation Mode shall be supported.

Network layer (layer 3)

The protocol IPv4 or IPv6 shall be used.

The address of the network layer or the DNS name shall be used to identify the BIP endpoint.

Transport layer (layer 4)

The protocol suites TCP and UDP shall be used.

Session, Presentation and Application layers (layers 5 to 7)

The protocols for communication with BIP endpoints are located in these layers. These are:

a) IP addresses and name services (see B.3 for IPv4 and B.5 for IPv6);

b) BIP messages – The protocol consists of the BIP exchanges specified in clause 12.

Time critical messages/Timeouts and error handling

Timing is determined by the related protocols of the respective layers. The handling of transmission and link errors shall be handled in layers 1 to 4.

B.3 Address and name setting in IPv4

B.3.1 A BIP endpoint obtains its network configuration data by one of the following mechanisms:

a) DHCP (see IETF RFC 2131);

b) dynamic configuration of IPv4 link-local address (see IETF RFC 3927);

c) static configuration.

B.3.2 The configuration data contains the following values:

a) the IPv4 address of the BIP endpoint;

b) the subnet mask of the local network;

c) gateway IPv4 address;

d) at least one DNS Server IP address.

NOTE – The broadcast address of the local network, which is necessary for the *Service Discovery Protocol* (see B.7), can be formed from the IP address and the subnet mask and therefore it does not need to be set.

B.3.3 When a DHCP server is used, the BIP endpoint obtains all necessary data either by static lists or dynamically. The in-BIP endpoint configurable DNS name can be sent from the BIP endpoint to a DHCP Server, which also provides the DNS-UPDATE extensions for co-operation with a configured DNS server (see IETF RFC 2136).

B.3.4 Dynamic configuration of IPv4 link-local address (IETF RFC 3927), also called Auto-IP is an ARP (see IETF RFC 826) based mechanism, which tries to find unique addresses without the help of a central service such as DHCP: Therefore an IPv4 address in the range 169.254.1.0 to 169.254.254.255 (see IETF RFC 3927) will be chosen with the help of a random number generator.

NOTE – IETF RFC 3927 also defines the timing of addressing trials and the behaviour in the case of address conflicts.

B.3.5 An administrator (via a management interface) does the static address configuration. The user interface has to provide the above-described data entries.

B.4 The network configuration function in IPv4

NOTE – See also Figure B.1.

B.4.1 A BIP endpoint is configured to use either static or dynamic network configuration. Which one is used is an implementation option. The factory values shall be specified in the operation manual as standard values. With static configuration, the administrator is responsible for the correctness of all settings.

NOTE – Wrong settings can mean that the BIP endpoint cannot be found via a network.

B.4.2 During dynamic configuration, the BIP endpoint first tries to get configuration data via a DHCP server. If none answers within a certain time, the BIP endpoint assumes that there is none available. After this exchange, the BIP endpoint uses the Auto-IP mechanism (see IETF RFC 3927) to set the configuration data itself. The Auto-IP mechanism defines that from time to time again, a DHCP server has to be searched.

NOTE – Error handling (especially the handling of address conflicts and hardware errors) is done in the TCP/IP stack or by the implementations of the DHCP or Auto-IP mechanisms.

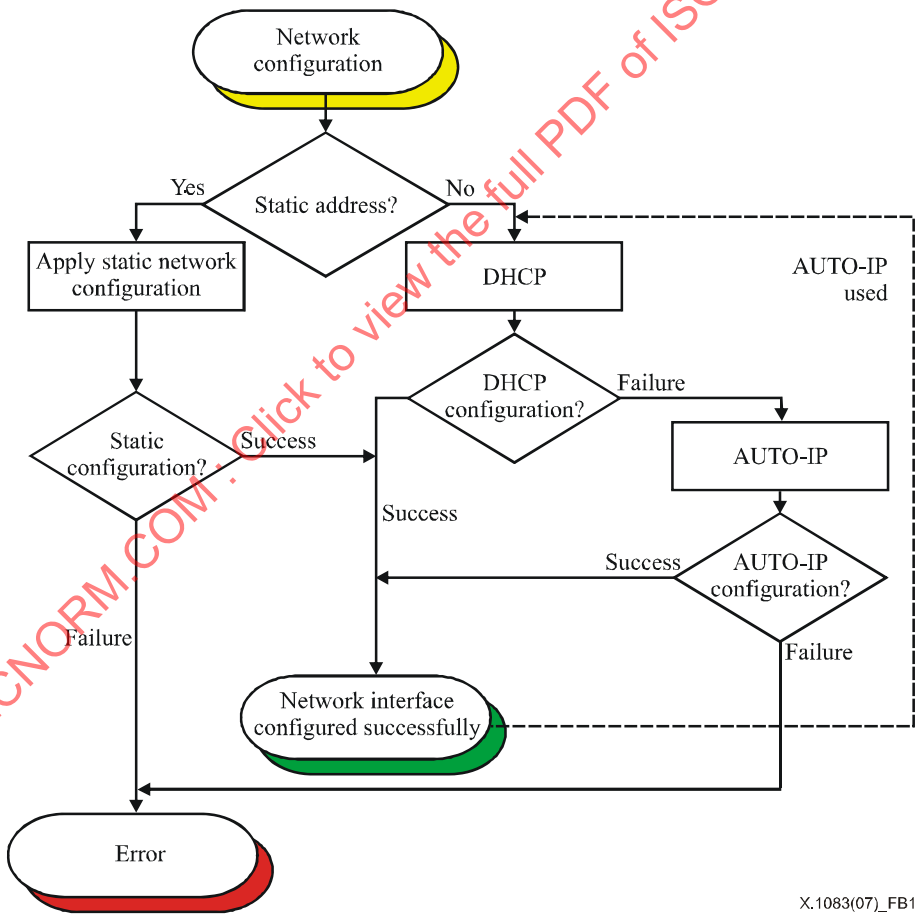

Figure B.1 – Network configuration function in IPv4

B.5 Address and name setting in IPv6

B.5.1 A BIP endpoint obtains its network configuration data by one of the following mechanisms:

a) Stateful autoconfiguration mechanism, DHCPv6 (see IETF RFC 3315);

b) Stateless autoconfiguration mechanism (see IETF RFC 2462);

c) static configuration.

B.5.2 The configuration data contains the following values:

a) the global unicast IPv6 address of the BIP endpoint;

b) the prefix length of the local network;

c) at least one DNS Server IPv6 address.

NOTE – The gateways addresses are obtained by the router discovery mechanism which is part of ICMPv6 (see IETF RFC 4443).

B.5.3 When stateful autoconfiguration is used, the BIP endpoint obtains all necessary data from a DHCPv6 server either by static lists or dynamically. The in-BIP endpoint configurable DNS name can be sent from the BIP endpoint to a DHCPv6 server (see IETF RFC 3315), which also provides the DNS-UPDATE extensions for co-operation with a configured DNS server (see IETF RFC 2136).

B.5.4 A stateless autoconfiguration mechanism should be used to obtain a global unicast IPv6 address when there is an IPv6 router on the network as follows. First the BIP configure a local-link IPv6 address derived from the hardware Ethernet address of the network interface. Then it sends a solicit message (see IETF RFC 3315) to detect the routers present on the network. If a router answers with an announcement message specifying that stateful shall be used, the BIP tries to use DHCPv6 (see IETF RFC 3315); if the answer allows stateless autoconfiguration, the BIP creates a global unicast IPv6 address which combines the IPv6 prefix received from the router and its own hardware Ethernet address (see IETF RFC 2462).

B.5.5 An administrator (via a BIP endpoint management interface) does the static address configuration. The user interface has to provide the entries described above.

B.6 The network configuration function in IPv6

NOTE – See also Figure B.2.

B.6.1 A BIP endpoint is configured to use either static or dynamic network configuration. Which one is used is an implementation option. The factory values shall be specified in the operation manual as standard values. With static configuration, the administrator is responsible for the correctness of all settings.

NOTE – Wrong settings can mean that the BIP endpoint cannot be found via a network.

B.6.2 During dynamic configuration, if stateful autoconfiguration has been specified, the BIP endpoint tries to get configuration data via a DHCPv6 server (see IETF RFC 3315). Otherwise, the BIP endpoint solicits router announcements. If it receives an announcement specifying stateful autoconfiguration, it tries to contact a DHCPv6 server; otherwise, it creates its own global unicast IPv6 address.

NOTE – Error handling (especially the handling of address conflicts and hardware errors) is done in the TCP/IP stack or by the implementations of autoconfiguration protocols.

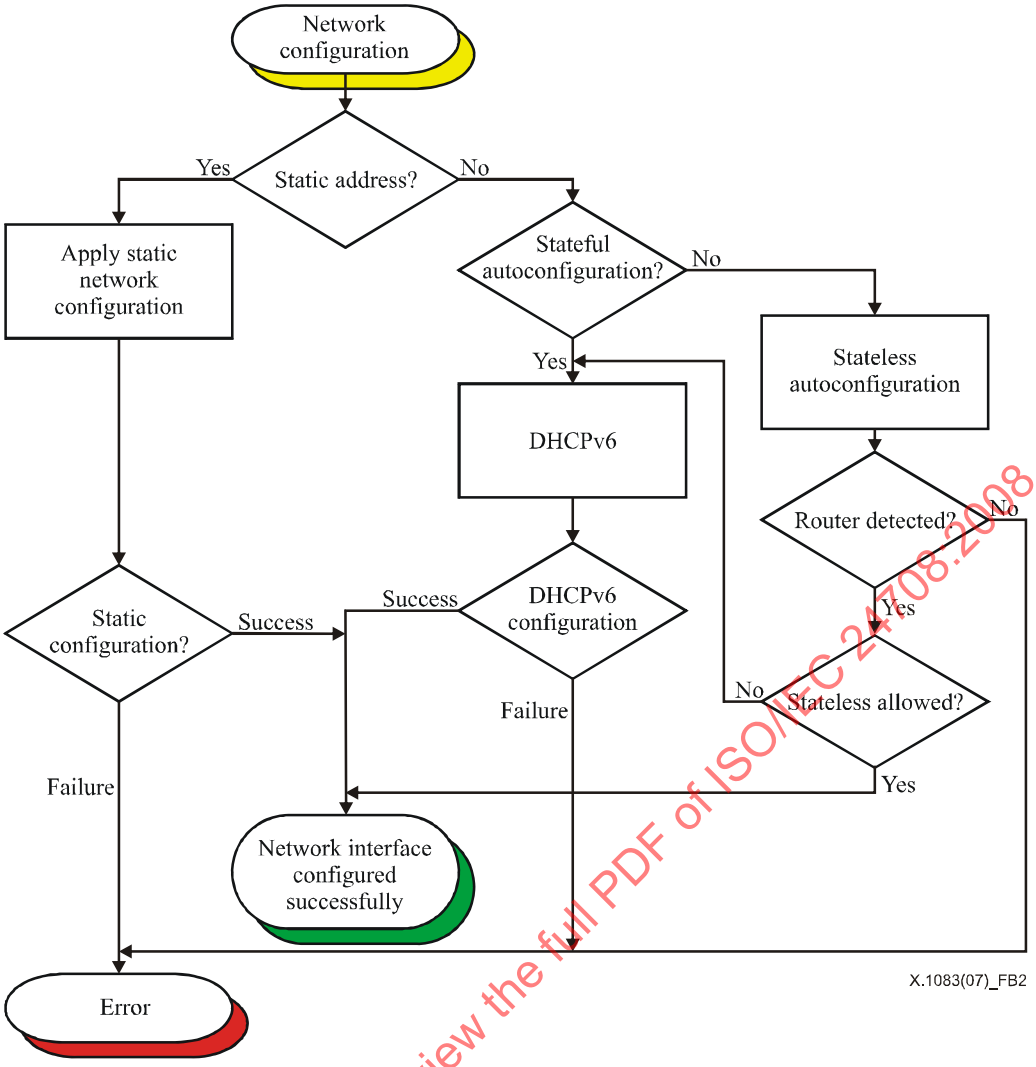

Figure B.2 – Network configuration function in IPv6

B.7 Discovery and announcement

The protocols for discovery of BIP endpoints are located in Session, Presentation and Application layers (layers 5 to 7). They allow automatic discovery of BIP slave endpoints by BIP master endpoints.

The use of a BIP endpoint in a network requires three areas to be addressed:

a) the address and name setting of the BIP endpoint: The BIP endpoint obtains an IP address and optionally a DNS name. Other network parameters will be set;

b) dynamic discovery of BIP endpoints by a BIP framework (Service Discovery): A broadcast or multicast based protocol enables applications to obtain the actual IP addresses and names of the endpoints. This step is performed in advance of the communication of BIP messages between applications;

c) Service Announcement: Additionally, BIP endpoints can introduce themselves to BIP master endpoints.

The BIP discovery messages (Service Discovery/Announcement) are transmitted via unprotected channels. When the BIP communication channel will be protected with one of the security protocols, the communication via BIP messages starts not before successful start-up of the security protocol. This way denial of service attacks can be avoided.

If there is manual configuration of BIP endpoints, there is no requirement to implement the service discovery and announcement (see B.8 and B.13). For dynamic discovery (see B.8) the service discovery protocol shall be implemented, but service announcement is optional.

The BIP slave endpoint has the option to display the BIP master endpoint for the support of security protocols in the field 'security protocols' in the service announcement package.

B.8 Service discovery

B.8.1 To discover a BIP endpoint in the network by a BIP framework, an UDP (see IETF RFC 768) based multicast (or broadcast in IPv4) protocol is used. The discovery protocol is not restricted to a local area network. This annex describes version 1.0 of the discovery protocol.

B.8.2 The service request is done by default via UDP port 4376 (service request port).

B.8.3 The BIP master endpoint sends the request as multicast (or broadcast in IPv4) transmission.

B.8.4 The BIP endpoint waits for an incoming service request on a socket configured to receive multicast messages.

NOTE – Therefore, it is only possible to receive those service requests where the transmission mode is equal to the receive mode of the BIP endpoint.

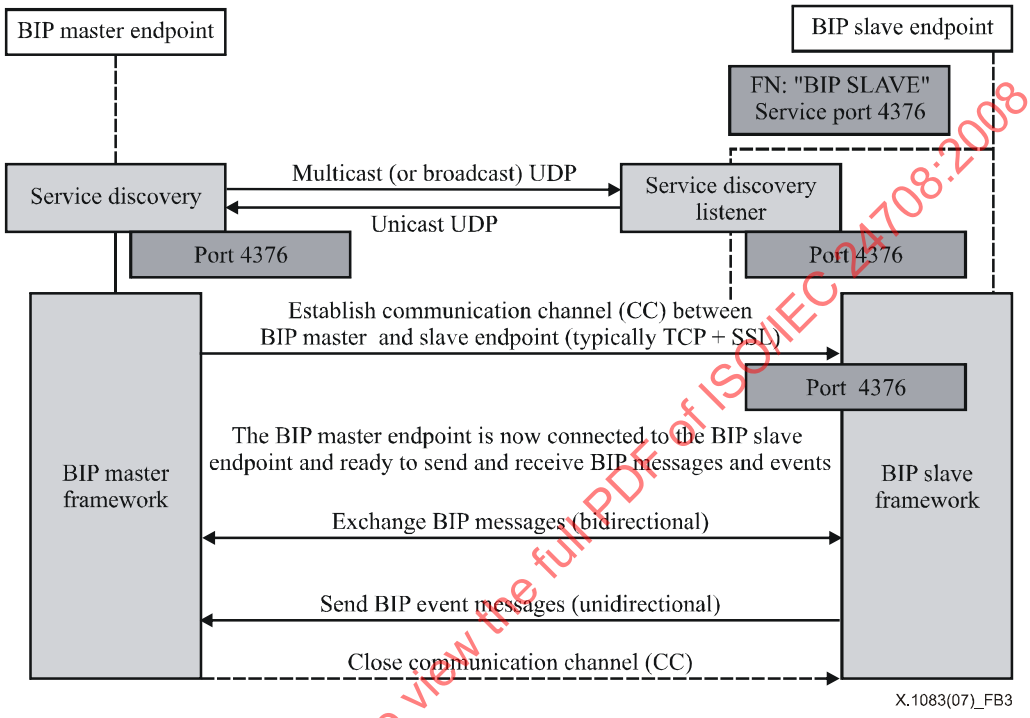

Figure B.3 – Principle communication channel schema

B.9 Service requests via broadcast (IPv4)

B.9.1 A broadcast can be sent as directed packets or as local packets. The target address controls the kind of broadcast:

a) a broadcast in the local network shall use the address 255.255.255.255;

b) for a directed broadcast, the address can be calculated from the IP address and the subnet mask of the BIP master endpoint.

B.9.2 For a directed broadcast into another subnet, the broadcast address has to be configurable. In addition, it must be ensured by the network administration that the broadcast will be routed to the target address and that reply packets can also be routed from the target address. The related configuration of the network components (router, firewall, etc.) is not in the scope of this Recommendation | International Standard.

B.9.3 When using broadcast, the transmitter does not normally receive its own packets. A BIP framework acting as both a master and a slave endpoint needs special handling in this case.

B.10 Service requests via multicast (IPv4 or IPv6)

When multicast is used, the multicast group address has to be substituted by the respective administration.

The network administration has to ensure the correct forwarding of BIP multicast messages between subnets.

B.11 Receiving service announcement packets

After transmission of a service request packet, the BIP master endpoint listens at port 4376 or at the given port in the service request for a service announcement packet from a BIP slave endpoint. The time to wait for a service announcement packet is not standardized.

It is a BIP master endpoint decision how long it waits for this type of message.

NOTE – To determine the waiting time, the transmission rate and usage of the networks have to be considered.

Each BIP endpoint receiving a service request packet shall send a service announcement package using UDP/IP to the master BIP endpoint: The target IP address and the target port shall be given in the service request packet, possibly as the default UDP port 4376.

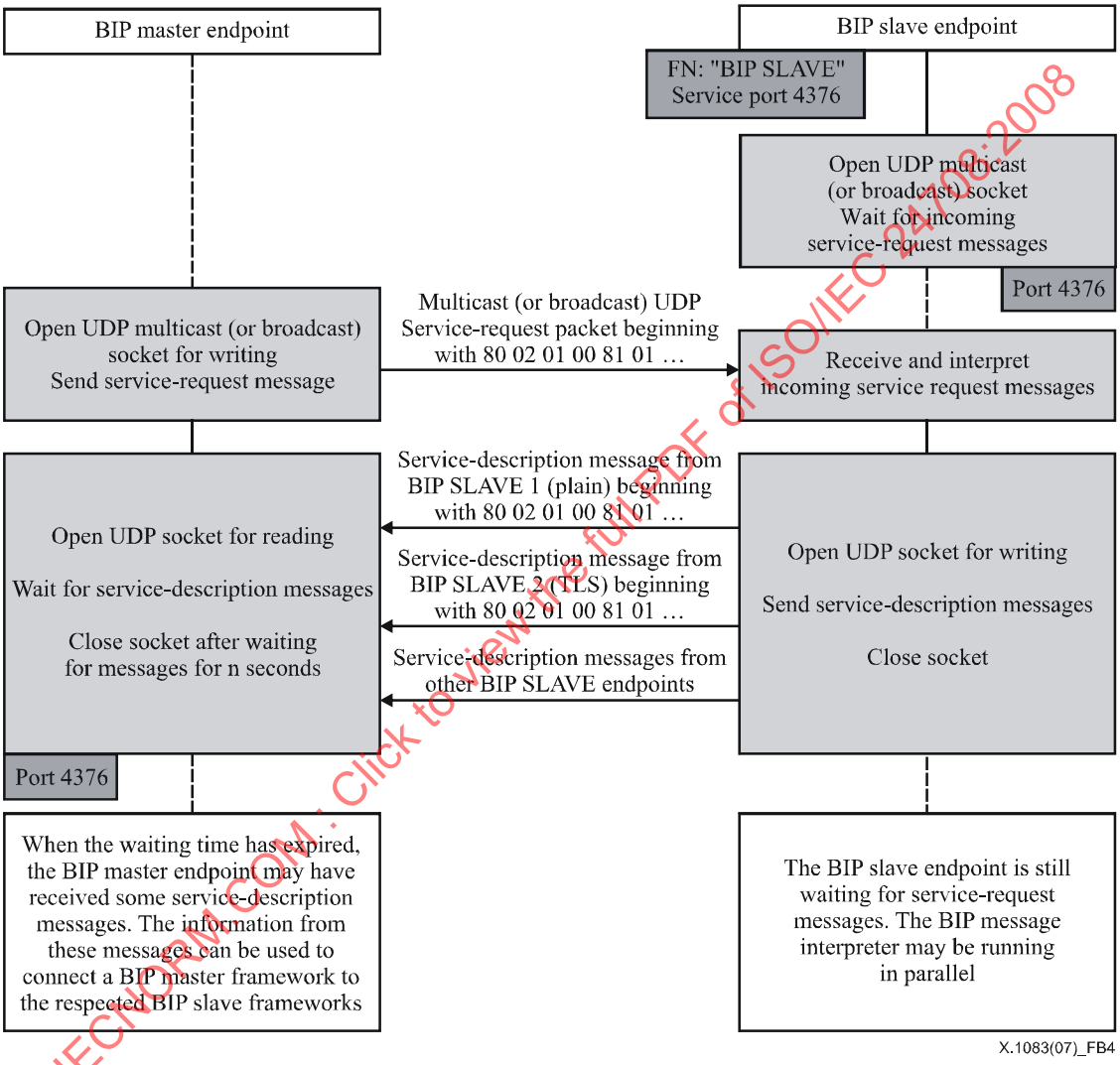

Figure B.4 – Service discovery broadcast/multicast

B.12 Format of discovery and announcement messages

The discovery and announcement messages are defined in the following ASN.1 module and have to encoded in Aligned PER (see ITU-T Rec. X.691 | ISO/IEC 8825-2).

```
BIP-DISCOVERY {joint-iso-itu-t bip(41) modules(0) bip-discovery(2) version1(1)}
DEFINITIONS AUTOMATIC TAGS ::=
BEGIN

Discovery  ::=  SEQUENCE {
        protocolVersion              ProtocolVersion,
        masterEndpointAddress        IPAddress,
        masterEndPort                Port  DEFAULT 4376,
        ...
        }

Announcement  ::=    SEQUENCE {
        protocolVersion              ProtocolVersion,
        slaveEndpointIPAddress       IPAddress,
        slaveEndpointMACAddress  MACAddress,
        slaveEndpointName   IA5String(SIZE(1..32)),
        bipMessagePort       Port  DEFAULT 4376,
        securityProtocols    SEQUENCE OF SecurityProtocol OPTIONAL,
        ...
        }

ProtocolVersion  ::=  SEQUENCE {
        major          INTEGER(0..255),
        minor          INTEGER(0..255)
        }

PAddress  ::=          CHOICE {
        ipv4           OCTET STRING(SIZE(4)),
        ipv6           OCTET STRING(SIZE(16))
        }

Port     ::=  INTEGER(0..65535)

MACAddress   ::=  OCTET STRING(SIZE(6))

SecurityProtocol    ::=SEQUENCE {
        id             SECURITY-PROTOCOL.&id({SecurityProtocols}),
        parameter      SECURITY-PROTOCOL.&Parameter({SecurityProtocols}{@id})
        }

SECURITY-PROTOCOL    ::=  CLASS {
        &id            OBJECT IDENTIFIER,
        &Parameter
        }

SecurityProtocols SECURITY-PROTOCOL ::= {...}

END
```

In an IPv4 network, the alternative **ipv4** of **IPAddress** shall be used, in an IPv6 network the **ipv6** of **IPAddress** shall be used.

The protocol shall be 1 for **major** component and 0 for **minor** component.

The default value of the port is the recommended port 4376. This port can be changed by use of the management interfaces.

NOTE – BIP master endpoints will be informed about the actual BIP message port in the service announcement packet.

The component **slaveEndpointMACAddress** is the address of the Ethernet interface of the BIP endpoint (see ISO/IEC TR 8802-1).

An announcement message can contain a list of security protocols supported by the slave endpoint. The absence of that list means no use of security in BIP transfers. Each security protocol is defined by an ASN.1 object identifier value and some parameter.

B.13 Service announcement

A BIP slave endpoint may introduce itself in the network by sending a service announcement prior to the receipt of a service request packet. To do that, it sends a packet to the port 4376 (service announcement port) using the broadcast address in IPv4 or the multicast address defined for service request in IPv4 or IPv6.

The master endpoints interested in the service announcement packets are waiting for these on the port 4376 and, in multicast mode, have to listen to the specified multicast address.

The BIP slave endpoint does not wait for an answer from a BIP master endpoint. The BIP slave endpoint should send service announcement packets after any change of configuration such as:

a) change of IP address;

b) change of BIP message port;

c) change of BIP slave endpoint name.

The autonomous transmission of service announcements allows a BIP master endpoint to detect changes in the network configuration without decreasing the interval to transmit service broadcast or multicast requests. Frequent transmission of service announcements should be avoided in order to reduce the network load.

B.14 Reset and restart

Changes configuration settings and some error conditions require the termination of BIP communications and the clearing of BIP endpoint internal states.

When the new settings are activated, to enable a faster restart, the BIP slave endpoint can also send service announcement messages.

At worst, the BIP master endpoint has to send service discovery requests until a BIP slave endpoint answers them.

The changes to configuration settings that require termination are:

a) change of an IP address;

b) change of the BIP message port;

c) change of the security settings for the BIP endpoint (including relevant certificates).

Other error conditions that require termination of the TCP/IP connection are specified in A.5.

After managed changes, both sides can terminate the connection in accordance with this part of the standard. After this termination, the new settings will be activated.

NOTE – When one of the described error states has been detected, the BIP slave endpoint shall shut down the network connection and all internal states shall be initialized.

B.15 Timing of the exchange of messages over a link channel

The BIP slave endpoint framework does not manage timings.

Master endpoints should take account of the fact that some operations may take a significant amount of time to complete.

The BIP master endpoint can request the status of the message processing by sending a service request message (see 16.17).

The BIP master endpoint can terminate messages by the Cancel message (see 16.57). The message termination can be necessary if:

a) the processing time exceeds the time calculated by the BIP master endpoint;

b) an error or abort condition has occurred at the BIP master endpoint side.

B.16 Security of the exchange of messages over a link channel

The BIP slave endpoint expects the protocol negotiation with the highest priorized protocol first. If the BIP master endpoint cannot establish a connection using this protocol, the second priorized will be taken.

This procedure proceeds until a common protocol could be found or no more negotiable protocol can be offered. If no protocol could be agreed on the TCP/IP, connection will be closed by the BIP slave endpoint.

Annex C

Specification of the SOAP/HTTP binding

(This annex forms an integral part of this Recommendation | International Standard)

C.1 General provisions

C.1.1 This annex specifies the use of BIP with the SOAP protocol using HTTP or HTTPS as a carrier.

C.1.2 A transport-level message for the binding specified in this annex shall consist of an HTTP message which carries a SOAP message encoded in XML 1.0, in accordance with the HTTP binding specified in W3C SOAP, clause 7.

C.1.3 The version of SOAP shall be either 1.1 or 1.2.

C.1.4 The version of HTTP shall be either 1.0 or 1.1. HTTP over TLS (HTTPS) may be used.

C.1.5 The HTTP SOAP Transmission Optimization Feature specified in W3C SOAP MTOM, clause 2, may be used.

NOTE – When this feature is used, any binary data (such as the BDB of a BIR) contained in a BIP message can be included in the HTTP message as a binary block and does not require the use of Base64.

C.1.6 A request BIP message shall be carried in an HTTP request message with the HTTP method POST. The body of the SOAP envelope within the HTTP request message shall contain a single occurrence of the global element **request** specified in C.4.1. The **soapAction** HTTP header (for SOAP 1.1) or the **action** parameter of the MIME media type (for SOAP 1.2) shall be set to "**oid:/BIP/Request**".

C.1.7 A response BIP message shall be carried in an HTTP response message returned by the slave endpoint in reply to the HTTP request message containing the corresponding request BIP message. The body of the SOAP envelope in the HTTP response message shall contain a single occurrence of the global element **response** specified in C.4.2.

C.1.8 A notification BIP message shall be carried in an HTTP request message with the HTTP method POST. The body of the SOAP envelope within the HTTP request message shall contain a single occurrence of the global element **notification** specified in C.4.3. The **soapAction** HTTP header (for SOAP 1.1) or the **action** parameter of the MIME media type (for SOAP 1.2) shall be set to "**oid:/BIP/Notification**".

C.1.9 An acknowledgement BIP message shall be carried in an HTTP response message returned by the master endpoint in reply to the HTTP request message containing the corresponding notification BIP message. The body of the SOAP envelope in the HTTP response message shall contain a single occurrence of the global element **acknowledgement** specified in C.4.4. For notification BIP messages that do not expect a corresponding acknowledgement BIP message, an HTTP response message with an empty body shall be sent by the master endpoint in response to the HTTP request.

C.1.10 The semantics of each of the XSD type definitions specified in this annex is defined by interpreting an element of that type as the EXTENDED-XER encoding (see ITU-T Rec. X.693/Amd.1 | ISO/IEC 8825-4/Amd.1) of an abstract value of an ASN.1 type that is specified in the referenced subclause of clause 16 or 17. The provisions of the referenced subclause indirectly apply to the creation and processing of elements of that XSD type through this semantic mapping.

C.2 Security considerations with SOAP/HTTP binding (tutorial)

There are three different ways of applying security techniques to a BIP message when using this binding:

a) point-to-point (carrier-level) security can be achieved by using HTTP over TLS (HTTPS) instead of regular HTTP over TCP/IP;

b) standard encryption and integrity techniques can be applied to the entire XML message or one or more portions of it (such as a BIR or a BDB in a BIR contained in the message) by using XML encryption and XML signature standards such as W3C XMLENC and W3C XMLDSIG; or

c) the BioAPI standard security provisions can be used to encrypt a BDB or to sign a BIR contained in a BIP message.

C.3 Schema header

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:bip="oid:/BIP"
  targetNamespace="oid:/BIP"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
```

C.4 Global elements

C.4.1 Global element request and type BIPRequest

```
<xs:element name="request" type="BIPRequest"/>

<xs:complexType name="BIPRequest">
  <xs:sequence>
      <xs:element name="masterEndpointIRI" type="EndpointIRI"/>
      <xs:element name="slaveEndpointIRI" type="EndpointIRI"/>
      <xs:element name="linkNumber" type="xs:unsignedInt"/>
      <xs:element name="requestId" type="xs:unsignedInt"/>
      <xs:choice>
       <xs:element name="addMaster"
            type="AddMaster-RequestParams"/>
       <xs:element name="deleteMaster"
            type="DeleteMaster-RequestParams"/>
       <xs:element name="bspLoad"
            type="BSPLoad-RequestParams"/>
       <xs:element name="bspUnload"
            type="BSPUnload-RequestParams"/>
       <xs:element name="queryUnits"
            type="QueryUnits-RequestParams"/>
       <xs:element name="queryBFPs"
            type="QueryBFPs-RequestParams"/>
       <xs:element name="bspAttach"
            type="BSPAttach-RequestParams"/>
       <xs:element name="bspDetach"
            type="BSPDetach-RequestParams"/>
       <xs:element name="enableUnitEvents"
            type="EnableUnitEvents-RequestParams"/>
       <xs:element name="enableEventNotifications"
            type="EnableEventNotifications-RequestParams"/>
       <xs:element name="controlUnit"
            type="ControlUnit-RequestParams"/>
       <xs:element name="control"
            type="Control-RequestParams"/>
       <xs:element name="freeBIRHandle"
            type="FreeBIRHandle-RequestParams"/>
       <xs:element name="getBIRFromHandle"
            type="GetBIRFromHandle-RequestParams"/>
       <xs:element name="getHeaderFromHandle"
            type="GetHeaderFromHandle-RequestParams"/>
       <xs:element name="subscribeToGUIEvents"
            type="SubscribeToGUIEvents-RequestParams"/>
       <xs:element name="unsubscribeFromGUIEvents"
            type="UnsubscribeFromGUIEvents-RequestParams"/>
       <xs:element name="redirectGUIEvents"
            type="RedirectGUIEvents-RequestParams"/>
       <xs:element name="unredirectGUIEvents"
            type="UnredirectGUIEvents-RequestParams"/>
       <xs:element name="queryGUIEventSubscriptions"
            type="QueryGUIEventSubscriptions-RequestParams"/>
       <xs:element name="notifyGUISelectEvent"
            type="NotifyGUISelectEvent-RequestParams"/>
       <xs:element name="notifyGUIStateEvent"
            type="NotifyGUIStateEvent-RequestParams"/>
       <xs:element name="notifyGUIProgressEvent"
            type="NotifyGUIProgressEvent-RequestParams"/>
       <xs:element name="capture"
            type="Capture-RequestParams"/>
       <xs:element name="createTemplate"
            type="CreateTemplate-RequestParams"/>
       <xs:element name="process"
            type="Process-RequestParams"/>
       <xs:element name="processWithAuxBIR"
            type="ProcessWithAuxBIR-RequestParams"/>
```

```
        <xs:element name="verifyMatch"
              type="VerifyMatch-RequestParams"/>
        <xs:element name="identifyMatch"
              type="IdentifyMatch-RequestParams"/>
        <xs:element name="enroll"
              type="Enroll-RequestParams"/>
        <xs:element name="verify"
              type="Verify-RequestParams"/>
        <xs:element name="identify"
              type="Identify-RequestParams"/>
        <xs:element name="import"
              type="Import-RequestParams"/>
        <xs:element name="presetIdentifyPopulation"
              type="PresetIdentifyPopulation-RequestParams"/>
        <xs:element name="transform"
              type="Transform-RequestParams"/>
        <xs:element name="dbOpen"
              type="DbOpen-RequestParams"/>
        <xs:element name="dbClose"
              type="DbClose-RequestParams"/>
        <xs:element name="dbCreate"
              type="DbCreate-RequestParams"/>
        <xs:element name="dbDelete"
              type="DbDelete-RequestParams"/>
        <xs:element name="dbSetMarker"
              type="DbSetMarker-RequestParams"/>
        <xs:element name="dbFreeMarker"
              type="DbFreeMarker-RequestParams"/>
        <xs:element name="dbStore"
              type="DbStoreBIR-RequestParams"/>
        <xs:element name="dbGetBIR"
              type="DbGetBIR-RequestParams"/>
        <xs:element name="dbGetNextBIR"
              type="DbGetNextBIR-RequestParams"/>
        <xs:element name="dbDeleteBIR"
              type="DbDeleteBIR-RequestParams"/>
        <xs:element name="calibrateSensor"
              type="CalibrateSensor-RequestParams"/>
        <xs:element name="setPowerMode"
              type="SetPowerMode-RequestParams"/>
        <xs:element name="setIndicatorStatus"
              type="SetIndicatorStatus-RequestParams"/>
        <xs:element name="getIndicatorStatus"
              type="GetIndicatorStatus-RequestParams"/>
        <xs:element name="cancel"
              type="Cancel-RequestParams"/>
        <xs:element name="registedBSP"
              type="RegisterBSP-RequestParams"/>
        <xs:element name="unregisterBSP"
              type="UnregisterBSP-RequestParams"/>
        <xs:element name="registerBFP"
              type="RegisterBFP-RequestParams"/>
        <xs:element name="unregisterBFP"
              type="UnregisterBFP-RequestParams"/>
      </xs:choice>
      <xs:any minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BIPRequest** specified in clause 14.

C.4.2 Global element response and type BIPResponse

```
<xs:element name="response" type="BIPResponse"/>

<xs:complexType name="BIPResponse">
  <xs:sequence>
    <xs:element name="slaveEndpointIRI" type="EndpointIRI"/>
    <xs:element name="masterEndpointIRI" type="EndpointIRI"/>
    <xs:element name="linkNumber" type="xs:unsignedInt"/>
    <xs:element name="requestId" type="xs:unsignedInt"/>
    <xs:choice>
     <xs:element name="addMaster"
           type="AddMaster-ResponseParams"/>
     <xs:element name="deleteMaster"
           type="DeleteMaster-ResponseParams"/>
```

```
<xs:element name="bspLoad"
      type="BSPLoad-ResponseParams"/>
<xs:element name="bspUnload"
      type="BSPUnload-ResponseParams"/>
<xs:element name="queryUnits"
      type="QueryUnits-ResponseParams"/>
<xs:element name="queryBFPs"
      type="QueryBFPs-ResponseParams"/>
<xs:element name="bspAttach"
      type="BSPAttach-ResponseParams"/>
<xs:element name="bspDetach"
      type="BSPDetach-ResponseParams"/>
<xs:element name="enableUnitEvents"
      type="EnableUnitEvents-ResponseParams"/>
<xs:element name="enableEventNotifications"
      type="EnableEventNotifications-ResponseParams"/>
<xs:element name="controlUnit"
      type="ControlUnit-ResponseParams"/>
<xs:element name="control"
      type="Control-ResponseParams"/>
<xs:element name="freeBIRHandle"
      type="FreeBIRHandle-ResponseParams"/>
<xs:element name="getBIRFromHandle"
      type="GetBIRFromHandle-ResponseParams"/>
<xs:element name="getHeaderFromHandle"
      type="GetHeaderFromHandle-ResponseParams"/>
<xs:element name="subscribeToGUIEvents"
      type="SubscribeToGUIEvents-ResponseParams"/>
<xs:element name="unsubscribeFromGUIEvents"
      type="UnsubscribeFromGUIEvents-ResponseParams"/>
<xs:element name="redirectGUIEvents"
      type="RedirectGUIEvents-ResponseParams"/>
<xs:element name="unredirectGUIEvents"
      type="UnredirectGUIEvents-ResponseParams"/>
<xs:element name="queryGUIEventSubscriptions"
      type="QueryGUIEventSubscriptions-ResponseParams"/>
<xs:element name="notifyGUISelectEvent"
      type="NotifyGUISelectEvent-ResponseParams"/>
<xs:element name="notifyGUIStateEvent"
      type="NotifyGUIStateEvent-ResponseParams"/>
<xs:element name="notifyGUIProgressEvent"
      type="NotifyGUIProgressEvent-ResponseParams"/>
<xs:element name="capture"
      type="Capture-ResponseParams"/>
<xs:element name="createTemplate"
      type="CreateTemplate-ResponseParams"/>
<xs:element name="process"
      type="Process-ResponseParams"/>
<xs:element name="processWithAuxBIR"
      type="ProcessWithAuxBIR-ResponseParams"/>
<xs:element name="verifyMatch"
      type="VerifyMatch-ResponseParams"/>
<xs:element name="identifyMatch"
      type="IdentifyMatch-ResponseParams"/>
<xs:element name="enroll"
      type="Enroll-ResponseParams"/>
<xs:element name="verify"
      type="Verify-ResponseParams"/>
<xs:element name="identify"
      type="Identify-ResponseParams"/>
<xs:element name="import"
      type="Import-ResponseParams"/>
<xs:element name="presetIdentifyPopulation"
      type="PresetIdentifyPopulation-ResponseParams"/>
<xs:element name="transform"
      type="Transform-ResponseParams"/>
<xs:element name="dbOpen"
      type="DbOpen-ResponseParams"/>
<xs:element name="dbClose"
      type="DbClose-ResponseParams"/>
<xs:element name="dbCreate"
      type="DbCreate-ResponseParams"/>
<xs:element name="dbDelete"
      type="DbDelete-ResponseParams"/>
<xs:element name="dbSetMarker"
      type="DbSetMarker-ResponseParams"/>
```

```
        <xs:element name="dbFreeMarker"
              type="DbFreeMarker-ResponseParams"/>
        <xs:element name="dbStore"
              type="DbStoreBIR-ResponseParams"/>
        <xs:element name="dbGetBIR"
              type="DbGetBIR-ResponseParams"/>
        <xs:element name="dbGetNextBIR"
              type="DbGetNextBIR-ResponseParams"/>
        <xs:element name="dbDeleteBIR"
              type="DbDeleteBIR-ResponseParams"/>
        <xs:element name="calibrateSensor"
              type="CalibrateSensor-ResponseParams"/>
        <xs:element name="setPowerMode"
              type="SetPowerMode-ResponseParams"/>
        <xs:element name="setIndicatorStatus"
              type="SetIndicatorStatus-ResponseParams"/>
        <xs:element name="getIndicatorStatus"
              type="GetIndicatorStatus-ResponseParams"/>
        <xs:element name="cancel"
              type="Cancel-ResponseParams"/>
        <xs:element name="registedBSP"
              type="RegisterBSP-ResponseParams"/>
        <xs:element name="unregisterBSP"
              type="UnregisterBSP-ResponseParams"/>
        <xs:element name="registerBFP"
              type="RegisterBFP-ResponseParams"/>
        <xs:element name="unregisterBFP"
              type="UnregisterBFP-ResponseParams"/>
      </xs:choice>
      <xs:element name="returnValue" type="BioAPI-RETURN"/>
      <xs:any minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BIPResponse** specified in clause 14.

C.4.3 Global element notification and type BIPNotification

```
  <xs:element name="notification" type="BIPNotification"/>

  <xs:complexType name="BIPNotification">
    <xs:sequence>
      <xs:element name="slaveEndpointIRI" type="EndpointIRI"/>
      <xs:element name="masterEndpointIRI" type="EndpointIRI"/>
      <xs:element name="linkNumber" type="xs:unsignedInt"/>
      <xs:element name="notificationId" type="xs:unsignedInt"/>
      <xs:choice>
       <xs:element name="masterDeletionEvent"
              type="MasterDeletionEvent-NotificationParams"/>
       <xs:element name="unitEvent"
              type="UnitEvent-NotificationParams"/>
       <xs:element name="guiSelectEvent"
              type="GUISelectEvent-NotificationParams"/>
       <xs:element name="guiStateEvent"
              type="GUIStateEvent-NotificationParams"/>
       <xs:element name="guiProgressEvent"
              type="GUIProgressEvent-NotificationParams"/>
       <xs:element name="bspRegistrationEvent"
              type="BSPRegistrationEvent-NotificationParams"/>
       <xs:element name="bspUnregistrationEvent"
              type="BSPUnregistrationEvent-NotificationParams"/>
       <xs:element name="bfpRegistrationEvent"
              type="BFPRegistrationEvent-NotificationParams"/>
       <xs:element name="bfpUnregistrationEvent"
              type="BFPUnregistrationEvent-NotificationParams"/>
      </xs:choice>
      <xs:any minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BIPNotification** specified in clause 14.

C.4.4 Global element acknowledgement and type BIPAcknowledgement

```
<xs:element name="acknowledgement" type="BIPAcknowledgement"/>

<xs:complexType name="BIPAcknowledgement">
  <xs:sequence>
     <xs:element name="masterEndpointIRI" type="EndpointIRI"/>
     <xs:element name="slaveEndpointIRI" type="EndpointIRI"/>
     <xs:element name="linkNumber" type="xs:unsignedInt"/>
     <xs:element name="notificationId" type="xs:unsignedInt"/>
     <xs:choice>
      <xs:element name="guiSelectEvent"
            type="GUISelectEvent-AcknowledgementParams"/>
      <xs:element name="guiStateEvent"
            type="GUIStateEvent-AcknowledgementParams"/>
      <xs:element name="guiProgressEvent"
            type="GUIProgressEvent-AcknowledgementParams"/>
     </xs:choice>
     <xs:element name="returnValue" type="BioAPI-RETURN"/>
     <xs:any minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BIPAcknowledgement** specified in clause 14.

C.5 Types

C.5.1 Type EndpointIRI

```
<xs:simpleType name="EndpointIRI">
  <xs:restriction base="xs:token"/>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **EndpointIRI** specified in 15.3.

C.5.2 Type BioAPI-BFP-LIST-ELEMENT

```
<xs:complexType name="BioAPI-BFP-LIST-ELEMENT">
  <xs:sequence>
     <xs:element name="category" type="BioAPI-CATEGORY"/>
     <xs:element name="bfpProductUuid" type="BioAPI-UUID"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BFP-LIST-ELEMENT** specified in 15.4.

C.5.3 Type BioAPI-BFP-SCHEMA

```
<xs:complexType name="BioAPI-BFP-SCHEMA">
  <xs:sequence>
     <xs:element name="bfpProductUuid" type="BioAPI-UUID"/>
     <xs:element name="category" type="BioAPI-CATEGORY"/>
     <xs:element name="description" type="BioAPI-STRING"/>
     <xs:element name="path" type="xs:string"/>
     <xs:element name="specVersion" type="BioAPI-VERSION"/>
     <xs:element name="productVersion" type="BioAPI-STRING"/>
     <xs:element name="vendor" type="BioAPI-STRING"/>
     <xs:element name="supportedFormats">
      <xs:complexType>
        <xs:sequence>
           <xs:element name="format" type="BioAPI-BIR-BIOMETRIC-DATA-FORMAT"
                 minOccurs="0" maxOccurs="4294967295"/>
        </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="factorsMask" type="BioAPI-BIR-BIOMETRIC-TYPE"/>
     <xs:element name="propertyUuid" type="BioAPI-UUID"/>
     <xs:element name="property" type="BioAPI-DATA"/>
     <xs:element name="hostingEndpointIRI" type="EndpointIRI"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BFP-SCHEMA** specified in 15.5.

C.5.4 Type BioAPI-BIR

```
<xs:complexType name="BioAPI-BIR">
  <xs:choice>
     <xs:element name="binaryBIR" type="xs:base64Binary"/>
     <xs:any namespace="##other" minOccurs="1" maxOccurs="1"/>
  </xs:choice>
  <xs:attribute name="patronFormatOwner" type="xs:unsignedShort" use="required"/>
  <xs:attribute name="patronFormatType" type="xs:unsignedShort" use="required"/>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BIR** specified in 15.6 with the following modification: the abstract value of the component **formattedBIR** (an octet string) may either be encoded in Base64 and wrapped in a **binaryBIR** element that is a child of the element of type **BioAPI-BIR**, or (only if the octet string is the UTF-8 encoding of an XML 1.0 element) may be directly included as a child of the element of type **BioAPI-BIR**.

NOTE – The use of the XML patron format specified in ISO/IEC 19785-3 is recommended.

C.5.5 Type BioAPI-BIR-ARRAY-POPULATION

```
<xs:complexType name="BioAPI-BIR-ARRAY-POPULATION">
  <xs:sequence>
     <xs:element name="members">
      <xs:complexType>
        <xs:sequence>
           <xs:element name="member" type="BioAPI-BIR"
                minOccurs="0" maxOccurs="4294967295"/>
        </xs:sequence>
      </xs:complexType>
     </xs:element>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BIR-ARRAY-POPULATION** specified in 15.7.

C.5.6 Type BioAPI-BIR-BIOMETRIC-DATA-FORMAT

```
<xs:complexType name="BioAPI-BIR-BIOMETRIC-DATA-FORMAT">
  <xs:sequence>
     <xs:element name="formatOwner" type="xs:unsignedShort"/>
     <xs:element name="formatType" type="xs:unsignedShort"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BIR-BIOMETRIC-DATA-FORMAT** specified in 15.8.

C.5.7 Type BioAPI-BIR-BIOMETRIC-PRODUCT-ID

```
<xs:complexType name="BioAPI-BIR-BIOMETRIC-PRODUCT-ID">
  <xs:sequence>
     <xs:element name="productOwner" type="xs:unsignedShort"/>
     <xs:element name="productType" type="xs:unsignedShort"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BIR-BIOMETRIC-PRODUCT-ID** specified in 15.9.

C.5.8 Type BioAPI-BIR-BIOMETRIC-TYPE

```
<xs:simpleType name="BioAPI-BIR-BIOMETRIC-TYPE">
  <xs:list itemType="SingleBiometricType"/>
</xs:simpleType>

<xs:simpleType name="SingleBiometricType">
  <xs:restriction base="xs:token">
      <xs:enumeration value="multipleBiometricTypes"/>
      <xs:enumeration value="face"/>
      <xs:enumeration value="voice"/>
      <xs:enumeration value="finger"/>
      <xs:enumeration value="iris"/>
      <xs:enumeration value="retina"/>
      <xs:enumeration value="handGeometry"/>
      <xs:enumeration value="signatureSign"/>
      <xs:enumeration value="keystroke"/>
      <xs:enumeration value="lipMovement"/>
      <xs:enumeration value="gait"/>
      <xs:enumeration value="vein"/>
      <xs:enumeration value="dna"/>
      <xs:enumeration value="ear"/>
      <xs:enumeration value="foot"/>
      <xs:enumeration value="scent"/>
      <xs:enumeration value="other"/>
      <xs:enumeration value="password"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BIR-BIOMETRIC-TYPE** specified in 15.10.

C.5.9 Type BioAPI-BIR-HANDLE

```
<xs:simpleType name="BioAPI-BIR-HANDLE">
  <xs:restriction base="xs:int"/>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BIR-HANDLE** specified in 15.12.

C.5.10 Type BioAPI-BIR-HEADER

```
<xs:complexType name="BioAPI-BIR-HEADER">
  <xs:choice>
      <xs:element name="binaryBIR" type="xs:base64Binary"/>
      <xs:any namespace="##other" minOccurs="1" maxOccurs="1"/>
  </xs:choice>
  <xs:attribute name="patronFormatOwner" type="xs:unsignedShort" use="required"/>
  <xs:attribute name="patronFormatType" type="xs:unsignedShort" use="required"/>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BIR-HEADER** specified in 15.13 with the following modification: the abstract value of the component **formattedBIR** (an octet string) may either be encoded in Base64 and wrapped in a **binaryBIR** element that is a child of the element of type **BioAPI-BIR-HEADER**, or (only if the octet string is the UTF-8 encoding of an XML 1.0 element) may be directly included as a child of the element of type **BioAPI-BIR-HEADER**.

NOTE – The use of the XML patron format specified in ISO/IEC 19785-3 is recommended.

C.5.11 Type BioAPI-BIR-PURPOSE

```
<xs:simpleType name="BioAPI-BIR-PURPOSE">
  <xs:restriction base="xs:token">
      <xs:enumeration value="verify"/>
      <xs:enumeration value="identify"/>
      <xs:enumeration value="enroll"/>
      <xs:enumeration value="enrollVerify"/>
      <xs:enumeration value="enrollIdentify"/>
      <xs:enumeration value="audit"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BIR-PURPOSE** specified in 15.14.

C.5.12 Type BioAPI-BIR-SECURITY-BLOCK-FORMAT

```
<xs:complexType name="BioAPI-BIR-SECURITY-BLOCK-FORMAT">
  <xs:sequence>
      <xs:element name="formatOwner" type="xs:unsignedShort"/>
      <xs:element name="formatType" type="xs:unsignedShort"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BIR-SECURITY-BLOCK-FORMAT** specified in 15.15.

C.5.13 Type BioAPI-BIR-SUBTYPE

```
<xs:complexType name="BioAPI-BIR-SUBTYPE">
  <xs:sequence>
      <xs:element name="subtype" type="SingleSubtype"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="SingleSubtype">
       <xs:choice>
          <xs:element name="anySubtype" type="anySubtype"/>
          <xs:element name="veinOnlySubtype" type="veinOnySubtype"/>
       </xs:choice>
      </xs:complexType>

      <xs:simpleType name="anySubtype">
  <xs:restriction base="xs:token">
      <xs:enumeration value="left"/>
      <xs:enumeration value="right"/>
      <xs:enumeration value="thumb"/>
      <xs:enumeration value="pointerFinger"/>
      <xs:enumeration value="middleFinger"/>
      <xs:enumeration value="ringFinger"/>
      <xs:enumeration value="littleFinger"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="veinOnlySubtype">
 <xs:restriction base="xs:token">
      <xs:enumeration value="veinPalm"/>
      <xs:enumeration value="veinBackofhand"/>
      <xs:enumeration value="veinWrist"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BIR-SUBTYPE** specified in 15.16.

C.5.14 Type BioAPI-BIR-SUBTYPE-MASK

```
<xs:simpleType name="BioAPI-BIR-SUBTYPE-MASK">
  <xs:list itemType="SingleSubtype-mask"/>
</xs:simpleType>

<xs:simpleType name="SingleSubtype-mask">
  <xs:restriction base="xs:token">
      <xs:enumeration value="left"/>
      <xs:enumeration value="right"/>
      <xs:enumeration value="leftThumb"/>
      <xs:enumeration value="leftPointerFinger"/>
      <xs:enumeration value="leftMiddleFinger"/>
      <xs:enumeration value="leftRingFinger"/>
      <xs:enumeration value="leftLittleFinger"/>
      <xs:enumeration value="rigthtThumb"/>
      <xs:enumeration value="rightPointerFinger"/>
      <xs:enumeration value="rightMiddleFinger"/>
      <xs:enumeration value="rightRingFinger"/>
      <xs:enumeration value="rightLittleFinger"/>
      <xs:enumeration value="left-vein-palm"/>
      <xs:enumeration value="left-vein-backofhand"/>
      <xs:enumeration value="left-vein-wrist"/>
      <xs:enumeration value="rigth-vein-palm"/>
      <xs:enumeration value="right-vein-backofhand"/>
      <xs:enumeration value="right-vein-wrist"/>
</xs:restriction>
  </xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BIR-SUBTYPE** specified in 15.17.

C.5.15 Type BioAPI-BSP-SCHEMA

```
<xs:complexType name="BioAPI-BSP-SCHEMA">
  <xs:sequence>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"/>
     <xs:element name="description" type="BioAPI-STRING"/>
     <xs:element name="path" type="xs:string" minOccurs="0"/>
     <xs:element name="specVersion" type="BioAPI-VERSION"/>
     <xs:element name="productVersion" type="BioAPI-STRING"/>
     <xs:element name="vendor" type="BioAPI-STRING"/>
     <xs:element name="supportedFormats">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="format" type="BioAPI-BIR-BIOMETRIC-DATA-FORMAT"
               minOccurs="0" maxOccurs="4294967295"/>
        </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="factorsMask" type="BioAPI-BIR-BIOMETRIC-TYPE"/>
     <xs:element name="operations" type="BioAPI-OPERATIONS-MASK"/>
     <xs:element name="options" type="BioAPI-OPTIONS-MASK"/>
     <xs:element name="payloadPolicy" type="BioAPI-FMR"/>
     <xs:element name="maxPayloadSize" type="xs:unsignedInt"/>
     <xs:element name="defaultVerifyTimeout" type="xs:int"/>
     <xs:element name="defaultIdentifyTimeout" type="xs:int"/>
     <xs:element name="defaultCaptureTimeout" type="xs:int"/>
     <xs:element name="defaultEnrollTimeout" type="xs:int"/>
     <xs:element name="defaultCalibrateTimeout" type="xs:int"/>
     <xs:element name="maxBSPDbSize" type="xs:unsignedInt"/>
     <xs:element name="maxIdentify" type="xs:unsignedInt"/>
     <xs:element name="hostingEndpointIRI" type="EndpointIRI"/>
     <xs:element name="bspAccessUuid" type="BioAPI-UUID" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-BSP-SCHEMA** specified in 15.19.

C.5.16 Type BioAPI-CANDIDATE

```
<xs:complexType name="BioAPI-CANDIDATE">
  <xs:sequence>
     <xs:choice>
      <xs:element name="birInDatabase" type="BioAPI-UUID"/>
      <xs:element name="birInArray" type="xs:unsignedInt"/>
      <xs:element name="birInPresetArray" type="xs:unsignedInt"/>
     </xs:choice>
     <xs:element name="fmrAchieved" type="BioAPI-FMR"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-CANDIDATE** specified in 15.20.

C.5.17 Type BioAPI-CATEGORY

```
<xs:simpleType name="BioAPI-CATEGORY">
  <xs:restriction base="xs:token">
     <xs:enumeration value="archive"/>
     <xs:enumeration value="comparisonAlgorithm"/>
     <xs:enumeration value="processingAlgorithm"/>
     <xs:enumeration value="sensor"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-CATEGORY** specified in 15.21.

C.5.18 Type BioAPI-DATA

```
<xs:simpleType name="BioAPI-DATA">
  <xs:restriction base="xs:base64Binary"/>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-DATA** specified in 15.22.

C.5.19 Type BioAPI-DATE

```
<xs:complexType name="BioAPI-DATE">
  <xs:sequence>
     <xs:element name="year" type="xs:unsignedShort"/>
     <xs:element name="month" type="xs:unsignedByte"/>
     <xs:element name="day" type="xs:unsignedByte"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-DATE** specified in 15.23.

C.5.20 Type BioAPI-DB-ACCESS-TYPE

```
<xs:simpleType name="BioAPI-DB-ACCESS-TYPE">
  <xs:list itemType="SingleAccessType"/>
</xs:simpleType>

<xs:simpleType name="SingleAccessType">
  <xs:restriction base="xs:token">
     <xs:enumeration value="read"/>
     <xs:enumeration value="write"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-DB-ACCESS-TYPE** specified in 15.24.

C.5.21 Type BioAPI-DB-MARKER-HANDLE

```
<xs:simpleType name="BioAPI-DB-MARKER-HANDLE">
  <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-DB-MARKER-HANDLE** specified in 15.25.

C.5.22 Type BioAPI-DB-HANDLE

```
<xs:simpleType name="BioAPI-DB-HANDLE">
  <xs:restriction base="xs:int"/>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-DB-HANDLE** specified in 15.26.

C.5.23 Type BioAPI-DBBIR-ID

```
<xs:complexType name="BioAPI-DBBIR-ID">
  <xs:sequence>
     <xs:element name="dbHandler" type="BioAPI-DB-HANDLE"/>
     <xs:element name="keyValue" type="BioAPI-UUID"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-DBBIR-ID** specified in 15.27.

C.5.24 Type BioAPI-DTG

```
<xs:complexType name="BioAPI-DTG">
  <xs:sequence>
     <xs:element name="date" type="BioAPI-DATE"/>
     <xs:element name="time" type="BioAPI-TIME"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-DTG** specified in 15.28.

C.5.25 Type BioAPI-UNIT-EVENT-TYPE

```
<xs:simpleType name="BioAPI-UNIT-EVENT-TYPE">
  <xs:restriction base="xs:token">
      <xs:enumeration value="insert"/>
      <xs:enumeration value="remove"/>
      <xs:enumeration value="fault"/>
      <xs:enumeration value="sourcePresent"/>
      <xs:enumeration value="sourceRemoved"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-UNIT-EVENT-TYPE** specified in 15.30.

C.5.26 Type BioAPI-UNIT-EVENT-TYPE-MASK

```
<xs:simpleType name="BioAPI-UNIT-EVENT-TYPE-MASK">
  <xs:list itemType="BioAPI-UNIT-EVENT-TYPE"/>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-UNIT-EVENT-TYPE-MASK** specified in 15.31.

C.5.27 Type BioAPI-FMR

```
<xs:simpleType name="BioAPI-FMR">
  <xs:restriction base="xs:int"/>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-FMR** specified in 15.32.

C.5.28 Type BioAPI-FRAMEWORK-SCHEMA

```
<xs:complexType name="BioAPI-FRAMEWORK-SCHEMA">
  <xs:sequence>
      <xs:element name="fwProductUuid" type="BioAPI-UUID"/>
      <xs:element name="description" type="BioAPI-STRING"/>
      <xs:element name="path" type="xs:string" minOccurs="0"/>
      <xs:element name="specVersion" type="BioAPI-VERSION"/>
      <xs:element name="productVersion" type="BioAPI-STRING"/>
      <xs:element name="vendor" type="BioAPI-STRING"/>
      <xs:element name="propertyUuid" type="BioAPI-UUID" minOccurs="0"/>
      <xs:element name="property" type="BioAPI-DATA" minOccurs="0"/>
      <xs:element name="hostingEndpointIRI" type="EndpointIRI"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-FRAMEWORK-SCHEMA** specified in 15.33.

C.5.29 Type BioAPI-GUI-BITMAP

```
<xs:complexType name="BioAPI-GUI-BITMAP">
  <xs:sequence>
      <xs:element name="subtypeMask" type="BioAPI-BIR-SUBTYPE-MASK"/>
      <xs:element name="width" type="xs:unsignedInt"/>
      <xs:element name="height" type="xs:unsignedInt"/>
      <xs:element name="bitmap" type="BioAPI-DATA" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-GUI-BITMAP** specified in 15.34.

C.5.30 Type BioAPI-GUI-BITMAP-ARRAY

```
<xs:complexType name="BioAPI-GUI-BITMAP-ARRAY">
  <xs:sequence>
     <xs:element name="members">
      <xs:complexType>
        <xs:sequence>
           <xs:element name="member" type="BioAPI-GUI-BITMAP"
                 minOccurs="0" maxOccurs="4294967295"/>
        </xs:sequence>
      </xs:complexType>
     </xs:element>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-GUI-BITMAP-ARRAY** specified in 15.35.

C.5.31 Type BioAPI-GUI-EVENT-SUBSCRIPTION

```
<xs:complexType name="BioAPI-GUI-EVENT-SUBSCRIPTION">
  <xs:sequence>
     <xs:element name="subscriberEndpointIRI" type="EndpointIRI"/>
     <xs:element name="guiEventSubscriptionUuid" type="BioAPI-UUID"/>
     <xs:element name="guiEventsSubscribed">
      <xs:simpleType>
        <xs:list itemType="SingleGUIEvent"/>
      </xs:simpleType>
     </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="SingleGUIEvent">
  <xs:restriction base="xs:token">
     <xs:enumeration value="select"/>
     <xs:enumeration value="state"/>
     <xs:enumeration value="progress"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-GUI-EVENT-SUBSCRIPTION** specified in 15.36.

C.5.32 Type BIOAPI-GUI-ENROLL-TYPE

```
<xs:simpleType name="BioAPI-GUI-ENROLL-TYPE">
  <xs:list>
     <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value="testVerify"/>
        <xs:enumeration value="multipleCapture"/>
      </xs:restriction>
     </xs:simpleType>
  </xs:list>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-ENROLL-TYPE** specified in 15.38.

C.5.33 Type BioAPI-GUI-MOMENT

```
<xs:simpleType name="BioAPI-GUI-MOMENT">
  <xs:restriction base="xs:token">
     <xs:enumeration value="beforeStart"/>
     <xs:enumeration value="during"/>
     <xs:enumeration value="afterEnd"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-GUI-MOMENT** specified in 15.37.

C.5.34 Type BioAPI-GUI-OPERATION

```
<xs:simpleType name="BioAPI-GUI-OPERATION">
  <xs:restriction base="xs:token">
      <xs:enumeration value="capture"/>
      <xs:enumeration value="process"/>
      <xs:enumeration value="createTemplate"/>
      <xs:enumeration value="verifyMatch"/>
      <xs:enumeration value="identifyMatch"/>
      <xs:enumeration value="verify"/>
      <xs:enumeration value="identify"/>
      <xs:enumeration value="enroll"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-GUI-OPERATION** specified in 15.39.

C.5.35 Type BioAPI-GUI-RESPONSE

```
<xs:simpleType name="BioAPI-GUI-RESPONSE">
  <xs:restriction base="xs:token">
      <xs:enumeration value="default"/>
      <xs:enumeration value="startCycle"/>
      <xs:enumeration value="startSubop"/>
      <xs:enumeration value="continueSubop"/>
      <xs:enumeration value="nextSubop"/>
      <xs:enumeration value="opComplete"/>
      <xs:enumeration value="abortSubop"/>
      <xs:enumeration value="recapture"/>
      <xs:enumeration value="restartCycle"/>
      <xs:enumeration value="cancelOp"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-GUI-RESPONSE** specified in 15.40.

C.5.36 Type BioAPI-GUI-SUBOPERATION

```
<xs:simpleType name="BioAPI-GUI-SUBOPERATION">
  <xs:restriction base="xs:token">
      <xs:enumeration value="capture"/>
      <xs:enumeration value="process"/>
      <xs:enumeration value="createTemplate"/>
      <xs:enumeration value="verifyMatch"/>
      <xs:enumeration value="identifyMatch"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-GUI-SUBOPERATION** specified in 15.41.

C.5.37 Type BioAPI-HANDLE

```
<xs:simpleType name="BioAPI-HANDLE">
  <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-HANDLE** specified in 15.42.

C.5.38 Type BioAPI-IDENTIFY-POPULATION

```
<xs:complexType name="BioAPI-IDENTIFY-POPULATION">
  <xs:choice>
      <xs:element name="birDataBase" type="BioAPI-DB-HANDLE"/>
      <xs:element name="birArray" type="BioAPI-BIR-ARRAY-POPULATION"/>
      <xs:element name="birPresetArray">
       <xs:complexType/>
      </xs:element>
  </xs:choice>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-IDENTIFY-POPULATION** specified in 15.43.

C.5.39 Type BioAPI-INDICATOR-STATUS

```
<xs:simpleType name="BioAPI-INDICATOR-STATUS">
  <xs:restriction base="xs:token">
     <xs:enumeration value="accept"/>
     <xs:enumeration value="reject"/>
     <xs:enumeration value="ready"/>
     <xs:enumeration value="busy"/>
     <xs:enumeration value="failure"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-INDICATOR-STATUS** specified in 15.45.

C.5.40 Type BioAPI-INPUT-BIR

```
<xs:complexType name="BioAPI-INPUT-BIR">
  <xs:choice>
     <xs:element name="birInDB" type="BioAPI-DBBIR-ID"/>
     <xs:element name="birInBSP" type="BioAPI-BIR-HANDLE"/>
     <xs:element name="bir" type="BioAPI-BIR"/>
  </xs:choice>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-INPUT-BIR** specified in 15.46.

C.5.41 Type BioAPI-OPERATIONS-MASK

```
<xs:simpleType name="BioAPI-OPERATIONS-MASK">
  <xs:list itemType="SingleOperation"/>
</xs:simpleType>

<xs:simpleType name="SingleOperation">
  <xs:restriction base="xs:token">
     <xs:enumeration value="enableEvents"/>
     <xs:enumeration value="subscribeToGUIEvents"/>
     <xs:enumeration value="capture"/>
     <xs:enumeration value="createTemplate"/>
     <xs:enumeration value="process"/>
     <xs:enumeration value="processWithAuxBIR"/>
     <xs:enumeration value="verifyMatch"/>
     <xs:enumeration value="identifyMatch"/>
     <xs:enumeration value="enroll"/>
     <xs:enumeration value="verify"/>
     <xs:enumeration value="identify"/>
     <xs:enumeration value="import"/>
     <xs:enumeration value="presetIdentifyPopulation"/>
     <xs:enumeration value="databaseOperations"/>
     <xs:enumeration value="setPowerMode"/>
     <xs:enumeration value="setIndicatorStatus"/>
     <xs:enumeration value="getIndicatorStatus"/>
     <xs:enumeration value="calibrateSensor"/>
     <xs:enumeration value="utilities"/>
     <xs:enumeration value="queryUnits"/>
     <xs:enumeration value="queryBFPs"/>
     <xs:enumeration value="controlUnit"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-OPERATIONS-MASK** specified in 15.48.

C.5.42 Type BioAPI-OPTIONS-MASK

```
<xs:simpleType name="BioAPI-OPTIONS-MASK">
  <xs:list itemType="SingleOption"/>
</xs:simpleType>

<xs:simpleType name="SingleOption">
  <xs:restriction base="xs:token">
      <xs:enumeration value="raw"/>
      <xs:enumeration value="qualityRaw"/>
      <xs:enumeration value="qualityIntermediate"/>
      <xs:enumeration value="qualityProcessed"/>
      <xs:enumeration value="appGUI"/>
      <xs:enumeration value="guiProgressEvents"/>
      <xs:enumeration value="sourcePresent"/>
      <xs:enumeration value="payload"/>
      <xs:enumeration value="birSign"/>
      <xs:enumeration value="birEncrypt"/>
      <xs:enumeration value="templateUpdate"/>
      <xs:enumeration value="adaptation"/>
      <xs:enumeration value="binning"/>
      <xs:enumeration value="selfContainedDevice"/>
      <xs:enumeration value="moc"/>
      <xs:enumeration value="subtypeToCapture"/>
      <xs:enumeration value="sensorBFP"/>
      <xs:enumeration value="archiveBFP"/>
      <xs:enumeration value="comparisonBFP"/>
      <xs:enumeration value="processingBFP"/>
      <xs:enumeration value="coarseScores"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-OPTIONS-MASK** specified in 15.49.

C.5.43 Type BioAPI-POWER-MODE

```
<xs:simpleType name="BioAPI-POWER-MODE">
  <xs:restriction base="xs:token">
      <xs:enumeration value="normal"/>
      <xs:enumeration value="detect"/>
      <xs:enumeration value="sleep"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-POWER-MODE** specified in 15.50.

C.5.44 Type BioAPI-RETURN

```
<xs:simpleType name="BioAPI-RETURN">
  <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-RETURN** specified in 15.52.

C.5.45 Type BioAPI-STRING

```
<xs:simpleType name="BioAPI-STRING">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-STRING** specified in 15.53.

C.5.46 Type BioAPI-TIME

```
<xs:complexType name="BioAPI-TIME">
  <xs:sequence>
      <xs:element name="hour" type="xs:unsignedByte"/>
      <xs:element name="minute" type="xs:unsignedByte"/>
      <xs:element name="second" type="xs:unsignedByte"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-TIME** specified in 15.54.

C.5.47 Type BioAPI-UNIT-ID

```
<xs:simpleType name="BioAPI-UNIT-ID">
  <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-UNIT-ID** specified in 15.55.

C.5.48 Type BioAPI-UNIT-LIST-ELEMENT

```
<xs:complexType name="BioAPI-UNIT-LIST-ELEMENT">
  <xs:sequence>
    <xs:element name="category" type="BioAPI-CATEGORY"/>
    <xs:element name="unitID" type="BioAPI-UNIT-ID"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-UNIT-LIST-ELEMENT** specified in 15.56.

C.5.49 Type BioAPI-UNIT-SCHEMA

```
<xs:complexType name="BioAPI-UNIT-SCHEMA">
  <xs:sequence>
    <xs:element name="bspProductUuid" type="BioAPI-UUID"/>
    <xs:element name="unitManagerProductUuid" type="BioAPI-UUID"/>
    <xs:element name="unitID" type="BioAPI-UNIT-ID"/>
    <xs:element name="category" type="BioAPI-CATEGORY"/>
    <xs:element name="unitProperties" type="BioAPI-UUID"/>
    <xs:element name="vendorInformation" type="BioAPI-STRING"/>
    <xs:element name="supportedUnitEvents" type="BioAPI-UNIT-EVENT-TYPE-MASK"/>
    <xs:element name="propertyUuid" type="BioAPI-UUID"/>
    <xs:element name="property" type="BioAPI-DATA"/>
    <xs:element name="hardwareVersion" type="BioAPI-STRING"/>
    <xs:element name="firmwareVersion" type="BioAPI-STRING"/>
    <xs:element name="softwareVersion" type="BioAPI-STRING"/>
    <xs:element name="hardwareSerialNumber" type="BioAPI-STRING"/>
    <xs:element name="authenticatedHardware" type="xs:boolean"/>
    <xs:element name="maxBSPDbSize" type="xs:unsignedInt"/>
    <xs:element name="maxIdentify" type="xs:unsignedInt"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-UNIT-SCHEMA** specified in 15.57.

C.5.50 Type BioAPI-UUID

```
<xs:simpleType name="BioAPI-UUID">
  <xs:restriction base="xs:token">
    <xs:length value="36"/>
  </xs:restriction>
</xs:simpleType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-UUID** specified in 15.58.

C.5.51 Type BioAPI-VERSION

```
<xs:complexType name="BioAPI-VERSION">
  <xs:sequence>
    <xs:element name="major" type="xs:unsignedByte"/>
    <xs:element name="minor" type="xs:unsignedByte"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BioAPI-VERSION** specified in 15.59.

C.6 Parameters of request BIP messages

C.6.1 Parameters of the addMaster request BIP message

```
<xs:complexType name="AddMaster-RequestParams">
  <xs:sequence>
     <xs:element name="bipVersion" type="BioAPI-VERSION"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **AddMaster-RequestParams** specified in 16.4.

C.6.2 Parameters of the deleteMaster request BIP message

```
<xs:complexType name="DeleteMaster-RequestParams"/>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **DeleteMaster-RequestParams** specified in 16.5.

C.6.3 Parameters of the bspLoad request BIP message

```
<xs:complexType name="BSPLoad-RequestParams">
  <xs:sequence>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"/>
     <xs:element name="unitEventSubscription" type="xs:boolean"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BSPLoad-RequestParams** specified in 16.9.

C.6.4 Parameters of the bspUnload request BIP message

```
<xs:complexType name="BSPUnload-RequestParams">
  <xs:sequence>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"/>
     <xs:element name="unitEventSubscription" type="xs:boolean"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BSPUnload-RequestParams** specified in 16.10.

C.6.5 Parameters of the queryUnits request BIP message

```
<xs:complexType name="QueryUnits-RequestParams">
  <xs:sequence>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **QueryUnits-RequestParams** specified in 16.11.

C.6.6 Parameters of the queryBFPs request BIP message

```
<xs:complexType name="QueryBFPs-RequestParams">
  <xs:sequence>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **QueryBFPs-RequestParams** specified in 16.12.

C.6.7 Parameters of the bspAttach request BIP message

```
<xs:complexType name="BSPAttach-RequestParams">
  <xs:sequence>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"/>
     <xs:element name="version" type="BioAPI-VERSION"/>
     <xs:element name="units">
      <xs:complexType>
        <xs:sequence>
           <xs:element name="unit" type="BioAPI-UNIT-LIST-ELEMENT"
                minOccurs="0" maxOccurs="4294967295"/>
        </xs:sequence>
      </xs:complexType>
     </xs:element>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BSPAttach-RequestParams** specified in 16.13.

C.6.8 Parameters of the bspDetach request BIP message

```
<xs:complexType name="BSPDetach-RequestParams">
  <xs:sequence>
     <xs:element name="originalBSPHandle" type="BioAPI-HANDLE"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **BSPDetach-RequestParams** specified in 16.14.

C.6.9 Parameters of the enableUnitEvents request BIP message

```
<xs:complexType name="EnableUnitEvents-RequestParams">
  <xs:sequence>
     <xs:element name="originalBSPHandle" type="BioAPI-HANDLE"/>
     <xs:element name="unitEvents" type="BioAPI-UNIT-EVENT-TYPE-MASK"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **EnableUnitEvents-RequestParams** specified in 16.15.

C.6.10 Parameters of the enableEventNotifications request BIP message

```
<xs:complexType name="EnableEventNotifications-RequestParams">
  <xs:sequence>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"/>
     <xs:element name="unitEventTypes" type="BioAPI-UNIT-EVENT-TYPE-MASK"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **EnableEventNotifications-RequestParams** specified in 16.16.

C.6.11 Parameters of the controlUnit request BIP message

```
<xs:complexType name="ControlUnit-RequestParams">
  <xs:sequence>
     <xs:element name="originalBSPHandle" type="BioAPI-HANDLE"/>
     <xs:element name="unitID" type="BioAPI-UNIT-ID"/>
     <xs:element name="controlCode" type="xs:unsignedInt"/>
     <xs:element name="inputData" type="BioAPI-DATA"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **ControlUnit-RequestParams** specified in 16.17.

C.6.12 Parameters of the controlRequest BIP message

```
<xs:complexType name="Control-RequestParams">
  <xs:sequence>
     <xs:element name="originalBSPHandle" type="BioAPI-HANDLE"/>
     <xs:element name="unitID" type="BioAPI-UNIT-ID"/>
     <xs:element name="controlCode" type="BioAPI-UUID"/>
     <xs:element name="inputData" type="BioAPI-DATA"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **Control-RequestParams** specified in 16.18.

C.6.13 Parameters of the freeBIRHandle request BIP message

```
<xs:complexType name="FreeBIRHandle-RequestParams">
  <xs:sequence>
     <xs:element name="originalBSPHandle" type="BioAPI-HANDLE"/>
     <xs:element name="birHandle" type="BioAPI-BIR-HANDLE"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **FreeBIRHandle-RequestParams** specified in 16.19.

C.6.14 Parameters of the getBIRFromHandle request BIP message

```
<xs:complexType name="GetBIRFromHandle-RequestParams">
  <xs:sequence>
     <xs:element name="originalBSPHandle" type="BioAPI-HANDLE"/>
     <xs:element name="birHandle" type="BioAPI-BIR-HANDLE"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **GetBIRFromHandle-RequestParams** specified in 16.20.

C.6.15 Parameters of the getHeaderFromHandle request BIP message

```
<xs:complexType name="GetHeaderFromHandle-RequestParams">
  <xs:sequence>
     <xs:element name="originalBSPHandle" type="BioAPI-HANDLE"/>
     <xs:element name="birHandle" type="BioAPI-BIR-HANDLE"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **GetHeaderFromHandle-RequestParams** specified in 16.21.

C.6.16 Parameters of the subscribeToGUIEvents request BIP message

```
<xs:complexType name="SubscribeToGUIEvents-RequestParams">
  <xs:sequence>
     <xs:element name="guiEventSubscriptionUuid" type="BioAPI-UUID"
        minOccurs="0"/>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"
        minOccurs="0"/>
     <xs:element name="originalBSPHandle" type="BioAPI-HANDLE"
        minOccurs="0"/>
     <xs:element name="guiSelectEventSubscribed" type="xs:boolean"/>
     <xs:element name="guiStateEventSubscribed" type="xs:boolean"/>
     <xs:element name="guiProgressEventSubscribed" type="xs:boolean"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **SubscribeToGUIEvents-RequestParams** specified in 16.22.

C.6.17 Parameters of the unsubscribeFromGUIEvents request BIP message

```
<xs:complexType name="UnsubscribeFromGUIEvents-RequestParams">
  <xs:sequence>
     <xs:element name="guiEventSubscriptionUuid" type="BioAPI-UUID"
        minOccurs="0"/>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"
        minOccurs="0"/>
     <xs:element name="originalBSPHandle" type="BioAPI-HANDLE"
        minOccurs="0"/>
     <xs:element name="guiSelectEventSubscribed" type="xs:boolean"/>
     <xs:element name="guiStateEventSubscribed" type="xs:boolean"/>
     <xs:element name="guiProgressEventSubscribed" type="xs:boolean"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **UnsubscribeFromGUIEvents-RequestParams** specified in 16.23.

C.6.18 Parameters of the queryGUIEventSubscriptions request BIP message

```
<xs:complexType name="QueryGUIEventSubscriptions-RequestParams">
  <xs:sequence>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **QueryGUIEventSubscriptions-RequestParams** specified in 16.24.

C.6.19 Parameters of the notifyGUISelectEvent request BIP message

```
<xs:complexType name="NotifyGUISelectEvent-RequestParams">
  <xs:sequence>
     <xs:element name="subscriberEndpointIRI" type="EndpointIRI"/>
     <xs:element name="guiEventSubscriptionUuid" type="BioAPI-UUID"/>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"/>
     <xs:element name="unitID" type="BioAPI-UNIT-ID"/>
     <xs:element name="enrollType" type="BioAPI-GUI-ENROLL-TYPE"/>
     <xs:element name="operation" type="BioAPI-GUI-OPERATION"/>
     <xs:element name="moment" type="BioAPI-GUI-MOMENT"/>
     <xs:element name="resultCode" type="BioAPI-RETURN"/>
     <xs:element name="maxNumEnrollSamples" type="xs:unsignedInt"/>
     <xs:element name="selectableInstances" type="BioAPI-BIR-SUBTYPE-MASK"/>
     <xs:element name="capturedInstances" type="BioAPI-BIR-SUBTYPE-MASK"/>
     <xs:element name="text" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **NotifyGUISelectEvent-RequestParams** specified in 16.25.

C.6.20 Parameters of the notifyGUIStateEvent request BIP message

```
<xs:complexType name="NotifyGUIStateEvent-RequestParams">
  <xs:sequence>
     <xs:element name="subscriberEndpointIRI" type="EndpointIRI"/>
     <xs:element name="guiEventSubscriptionUuid" type="BioAPI-UUID"/>
     <xs:element name="bspProductUuid" type="BioAPI-UUID"/>
     <xs:element name="unitID" type="BioAPI-UNIT-ID"/>
     <xs:element name="operation" type="BioAPI-GUI-OPERATION"/>
     <xs:element name="suboperation" type="BioAPI-GUI-SUBOPERATION"/>
     <xs:element name="purpose" type="BioAPI-BIR-PURPOSE"/>
     <xs:element name="moment" type="BioAPI-GUI-MOMENT"/>
     <xs:element name="resultCode" type="BioAPI-RETURN"/>
     <xs:element name="enrollSampleIndex" type="xs:int"/>
     <xs:element name="bitmaps" type="BioAPI-GUI-BITMAP-ARRAY" minOccurs="0"/>
     <xs:element name="text" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

The semantics and the use of this XSD type are defined by interpreting an element of this type as the EXTENDED-XER encoding of an abstract value of the ASN.1 type **NotifyGUIStateEvent-RequestParams** specified in 16.26.