BRITISH STANDARD

BS ISO/IEC 24708:2008

Information technology — Biometrics — BioAPI Interworking Protocol

ICS 35.040

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW

Documents
Document Sharing Hub
British Standards

Full document details: BS ISO/IEC 24708:2008

National foreword

This British Standard is the UK implementation of ISO/IEC 24708:2008.

The UK participation in its preparation was entrusted to Technical Committee IST/44, Biometrics.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 April 2009.

© BSI 2009

ISBN 978 0 580 54245 9

Amendments/corrigenda issued since publication

Date	Comments

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 24708:2008

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)

Documents
Document Sharing Hub

© ISO/IEC 2008

Full document details: BS ISO/IEC 24708:2008

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2008

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

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

© ISO/IEC 2008 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 24708:2008

CONTENTS

Page

1 Scope 1
2 Normative references 2
2.1 Identical Recommendations | International Standards 2
2.2 Paired Recommendations | International Standards equivalent in technical content 2
2.3 Additional references 2
3 Conformance 3
4 Abbreviations 4
5 Conventions 5
6 Conformance 5
7 BIP architecture 7
7.1 BIP-enabled frameworks 7
7.2 BIP messages 8
7.3 BIP endpoints 8
7.4 BIP links 9
7.5 Transport protocol bindings 10
7.6 Creation and destruction of BIP links 10
8 Remote GUI event notifications 11
9 Examples of possible system configurations 12
10 BIR format 15
11 Identification of BIP endpoints, applications, and BSPs 15
12 Overview of BIP exchanges 16
12.1 Security and privacy provisions 16
12.2 Application invocation of functions on a remote BSP 16
12.3 Application invocation of functions with no associated BIP message 18
12.4 Event notifications 18
13 General provisions 18
14 BIP message syntax 21
15 BioAPI and BIP types 24
15.1 Integers 24
15.2 Character strings 25
15.3 Uniform resource identifiers designating BIP endpoints 25
15.4 Type BioAPI_BFP_LIST_ELEMENT 26
15.5 Type BioAPI_BFP_SCHEMA 26
15.6 Type BioAPI_BIR 28
15.7 Type BioAPI_BIR_ARRAY_POPULATION 28
15.8 Type BioAPI_BIR_BIOMETRIC_DATA_FORMAT 29
15.9 Type BioAPI_BIR_BIOMETRIC_PRODUCT_ID 29
15.10 Type BioAPI_BIR_BIOMETRIC_TYPE 29
15.11 Type BioAPI_BIR_DATA_TYPE 30
15.12 Type BioAPI_BIR_HANDLE 31
15.13 Type BioAPI_BIR_HEADER 31
15.14 Type BioAPI_BIR_PURPOSE 32
15.15 Type BioAPI_BIR_SECURITY_BLOCK_FORMAT 33
15.16 Type BioAPI_BIR_SUBTYPE 33
15.17 Type BioAPI_BIR_SUBTYPE_MASK 34
15.18 Type BioAPI_BOOL 35
15.19 Type BioAPI_BSP_SCHEMA 35
15.20 Type BioAPI_CANDIDATE 37
15.21 Type BioAPI_CATEGORY 38

Documents
Document Sharing Hub

© ISO/IEC 2008 – All rights reserved

Full document details: BS ISO/IEC 24708:2008

Page

15.22 Type BioAPI_DATA 38
15.23 Type BioAPI_DATE 39
15.24 Type BioAPI_DB_ACCESS_TYPE 39
15.25 Type BioAPI_DB_MARKER_HANDLE 40
15.26 Type BioAPI_DB_HANDLE 40
15.27 Type BioAPI_DBBIR_ID 40
15.28 Type BioAPI_DTG 41
15.29 Type BioAPI_ERROR_INFO 41
15.30 Type BioAPI_EVENT 41
15.31 Type BioAPI_EVENT_MASK 42
15.32 Type BioAPI_FMR 42
15.33 Type BioAPI_FRAMEWORK_SCHEMA 42
15.34 Type BioAPI_GUI_BITMAP 43
15.35 Type BioAPI_GUI_BITMAP_ARRAY 44
15.36 Type BioAPI_GUI_EVENT_SUBSCRIPTION 44
15.37 Type BioAPI_GUI_MOMENT 45
15.38 Type BioAPI_GUI_ENROLL_TYPE 45
15.39 Type BioAPI_GUI_OPERATION 45
15.40 Type BioAPI_GUI_RESPONSE 46
15.41 Type BioAPI_GUI_SUBOPERATION 47
15.42 Type BioAPI_HANDLE 48
15.43 Type BioAPI_IDENTIFY_POPULATION 48
15.44 Type BioAPI_IDENTIFY_POPULATION_TYPE 49
15.45 Type BioAPI_INDICATOR_STATUS 49
15.46 Type BioAPI_INPUT_BIR 50
15.47 Type BioAPI_INPUT_BIR_FORM 50
15.48 Type BioAPI_OPERATIONS_MASK 51
15.49 Type BioAPI_OPTIONS_MASK 52
15.50 Type BioAPI_POWER_MODE 53
15.51 Type BioAPI_QUALITY 53
15.52 Type BioAPI_RETURN 53
15.53 Type BioAPI_STRING 53
15.54 Type BioAPI_TIME 54
15.55 Type BioAPI_UNIT_ID 54
15.56 Type BioAPI_UNIT_LIST_ELEMENT 55
15.57 Type BioAPI_UNIT_SCHEMA 55
15.58 Type BioAPI_UUID 56
15.59 Type BioAPI_VERSION 56

16 Functions defined in BioAPI and corresponding BIP messages 57
16.1 Function BioAPI_Init 57
16.2 Function BioAPI_InitEndpoint 58
16.3 Function BioAPI_Terminate 58
16.4 Function BioAPI_LinkToEndpoint 59
16.5 Function BioAPI_UnlinkFromEndpoint 62
16.6 Function BioAPI_EnumFrameworks 63
16.7 Function BioAPI_EnumBSPs 63
16.8 Function BioAPI_EnumBFPs 64
16.9 Function BioAPI_BSPLoad 65
16.10 Function BioAPI_BSPUnload 67
16.11 Function BioAPI_QueryUnits 69
16.12 Function BioAPI_QueryBFPs 70
16.13 Function BioAPI_BSPAttach 72
16.14 Function BioAPI_BSPDetach 75

Documents
Document Sharing Hub

© ISO/IEC 2008 – All rights reserved

Full document details: BS ISO/IEC 24708:2008

Page
16.15 Function BioAPI_EnableEvents 76
16.16 Function BioAPI_EnableEventNotifications 77
16.17 Function BioAPI_ControlUnit 79
16.18 Function BioAPI_Control 80
16.19 Function BioAPI_FreeBIRHandle 81
16.20 Function BioAPI_GetBIRFromHandle 82
16.21 Function BioAPI_GetHeaderFromHandle 83
16.22 Function BioAPI_SubscribeToGUIEvents 84
16.23 Function BioAPI_UnsubscribeFromGUIEvents 88
16.24 Function BioAPI_QueryGUIEventSubscriptions 92
16.25 Function BioAPI_NotifyGUISelectEvent 94
16.26 Function BioAPI_NotifyGUIStateEvent 96
16.27 Function BioAPI_NotifyGUIProgressEvent 99
16.28 Function BioAPI_RedirectGUIEvents 101
16.29 Function BioAPI_UnredirectGUIEvents 102
16.30 Function BioAPI_Capture 104
16.31 Function BioAPI_CreateTemplate 105
16.32 Function BioAPI_Process 106
16.33 Function BioAPI_ProcessWithAuxBIR 107
16.34 Function BioAPI_VerifyMatch 109
16.35 Function BioAPI_IdentifyMatch 110
16.36 Function BioAPI_Enroll 112
16.37 Function BioAPI_Verify 113
16.38 Function BioAPI_Identify 115
16.39 Function BioAPI_Import 116
16.40 Function BioAPI_PresetIdentifyPopulation 118
16.41 Function BioAPI_Transform 118
16.42 Function BioAPI_DbOpen 120
16.43 Function BioAPI_DbClose 121
16.44 Function BioAPI_DbCreate 122
16.45 Function BioAPI_DbDelete 123
16.46 Function BioAPI_DbSetMarker 124
16.47 Function BioAPI_DbFreeMarker 124
16.48 Function BioAPI_DbStoreBIR 125
16.49 Function BioAPI_DbGetBIR 126
16.50 Function BioAPI_DbGetNextBIR 127
16.51 Function BioAPI_DbDeleteBIR 128
16.52 Function BioAPI_CalibrateSensor 129
16.53 Function BioAPI_SetPowerMode 130
16.54 Function BioAPI_SetIndicatorStatus 131
16.55 Function BioAPI_GetIndicatorStatus 132
16.56 Function BioAPI_GetLastErrorInfo 133
16.57 Function BioAPI_Cancel 133
16.58 Function BioAPI_Free 134
16.59 Function BioAPI_RegisterBSP 134
16.60 Function BioAPI_UnregisterBSP 136
16.61 Function BioAPI_RegisterBFP 138
16.62 Function BioAPI_UnregisterBFP 140
17 Callback functions defined in BioAPI and corresponding BIP messages 142
17.1 Callback function BioAPI_EVENT_HANDLER 142
17.2 Callback function BioAPI_GUI_SELECT_EVENT_HANDLER 144
17.3 Callback function BioAPI_GUI_STATE_EVENT_HANDLER 148
17.4 Callback function BioAPI_GUI_PROGRESS_EVENT_HANDLER 152

Documents
Document Sharing Hub

© ISO/IEC 2008 – All rights reserved

Full document details: BS ISO/IEC 24708:2008

Page

18 Conceptual tables 156
18.1 The MasterEndpoints conceptual table 156
18.2 The VisibleEndpoints conceptual table 158
18.3 The VisibleBSPRegistrations conceptual table 158
18.4 The VisibleBFPRegistrations conceptual table 161
18.5 The RunningBSPLocalReferences conceptual table 163
18.6 The RunningBSPRemoteReferences conceptual table 164
18.7 The UnitEventNotificationDisablers conceptual table 165
18.8 The AttachSessionLocalReferences conceptual table 166
18.9 The AttachSessionRemoteReferences conceptual table 168
18.10 The GUIEventLocalSubscriptions conceptual table 169
18.11 The GUIEventRemoteSubscriptions conceptual table 171
18.12 The GUIEventRedirectors conceptual table 172
18.13 The ApplicationOwnedMemoryBlocks conceptual table 173
19 Converting between a C pointer variable and a corresponding ASN.1 component (1) 174
20 Converting between a C pointer variable and a corresponding ASN.1 component (2) 174
21 Converting between a C pointer variable and a corresponding ASN.1 component (3) 175
22 Initializing and checking a C pointer variable having no corresponding ASN.1 component 175
23 Determining a hosting endpoint and a BSP product UUID from a BSP UUID 175
24 Determining a hosting endpoint and an original BSP handle from a local BSP handle 176
25 Converting BSP UUIDs 176
26 Converting BSP handles 176
27 Processing an incoming function call by exchanging a request/response BIP message pair with a slave endpoint 176
28 Processing an incoming request BIP message via an internal BioAPI function call 177
29 Notifying a unit event to zero or more subscribers 177
30 Notifying a GUI select event to a subscriber 178
31 Notifying a GUI state event to a subscriber 180
32 Notifying a GUI progress event to a subscriber 181
33 Handling unconvertible C values 183
Annex A – Specification of the TCP/IP binding 184
A.1 General 184
A.2 Transport-level message 184
A.3 TCP/IP connection between two BIP endpoints 185
A.4 Role of endpoint 185
A.5 Closing the connection on errors 186
A.6 Transport of BIP messages 186
A.7 Usage of IRIs 186
Annex B – Specification of discovery and announcement in TCP/IP binding 187
B.1 General 187
B.2 The PnP mechanisms 187
B.3 Address and name setting in IPv4 187
B.4 The network configuration function in IPv4 188
B.5 Address and name setting in IPv6 189
B.6 The network configuration function in IPv6 189
B.7 Discovery and announcement 190
B.8 Service discovery 191
B.9 Service requests via broadcast (IPv4) 191
B.10 Service requests via multicast (IPv4 or IPv6) 191
B.11 Receiving service announcement packets 192

Documents
Document Sharing Hub

© ISO/IEC 2008 – All rights reserved

Full document details: BS ISO/IEC 24708:2008

Page
B.12 Format of discovery and announcement messages 193
B.13 Service announcement 194
B.14 Reset and restart 194
B.15 Timing of the exchange of messages over a link channel 194
B.16 Security of the exchange of messages over a link channel 194
Annex C – Specification of the SOAP/HTTP binding 195
C.1 General provisions 195
C.2 Security considerations with SOAP/HTTP binding (tutorial) 195
C.3 Schema header 196
C.4 Global elements 196
C.5 Types 200
C.6 Parameters of request BIP messages 212
C.7 Parameters of response BIP messages 223
C.8 Parameters of notification BIP messages 231
C.9 Parameters of acknowledgement BIP messages 233
C.10 Closure of the schema 233
C.11 Example 233
Annex D – Clarification of minimal requirements for simple systems 244
D.1 A simple system with a single fixed biometric device 244
D.2 A simple system with a single database of BIRs 244
Annex E – Possible scenarios involving the use of the BioAPI interworking protocol 245
E.1 Access to a central national database for security and health administration 245
E.2 Registration of individuals at a point of entry, or a local registration centre 245
E.3 Theme-park access 245
Annex F – Formal ASN.1 modules 246
Annex G – Bibliography 270

© ISO/IEC 2008 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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.

© ISO/IEC 2008 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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.

© ISO/IEC 2008 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 24708:2008

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.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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*.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

– 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.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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;

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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).

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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;

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

– 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

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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).

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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.

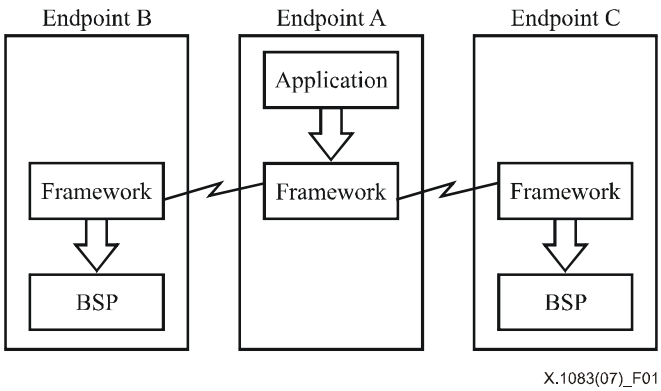

Figure 1 – An application communicating with two other BSPs on two remote systems

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

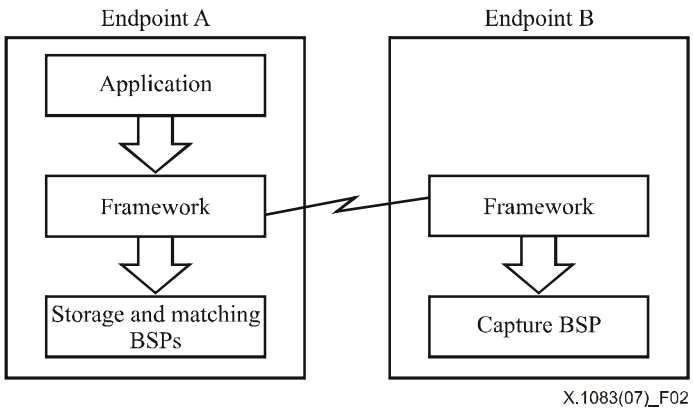

Figure 2 – An application communicating with a remote capture BSP, with all other functions performed locally

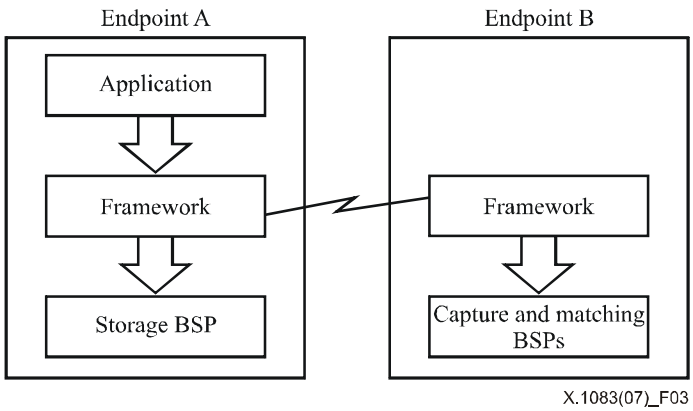

Figure 3 – An application communicating with a remote capture BSP, with remote matching and local storage

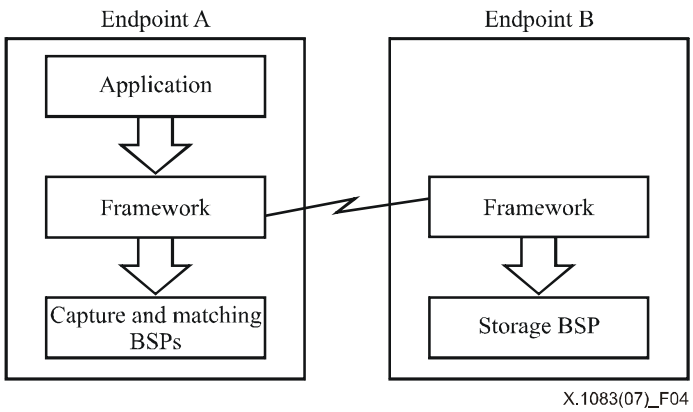

Figure 4 – An application communicating with a remote storage BSP, with all other functions performed locally

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

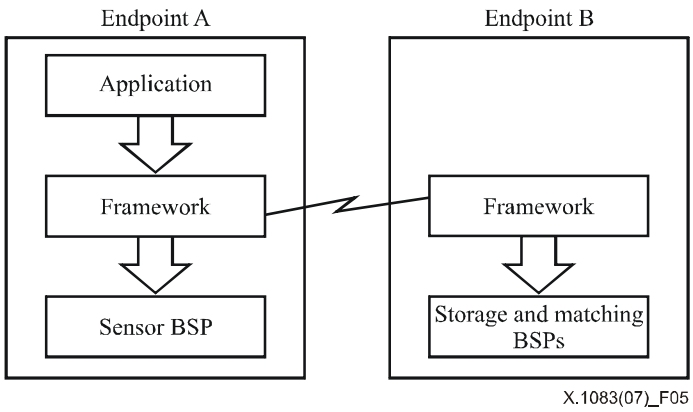

Figure 5 – An application communicating with a remote storage and matching BSP, with a local sensor

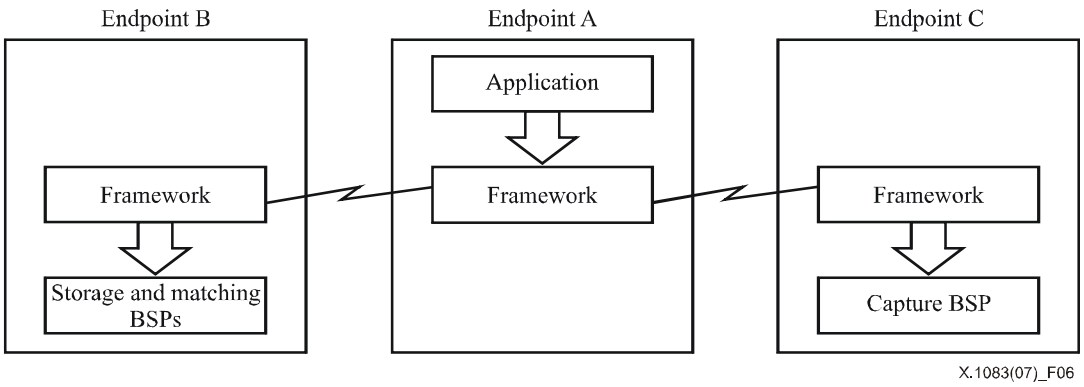

Figure 6 – An application communicating with two remote BSPs, one providing storage and matching and the other providing capture

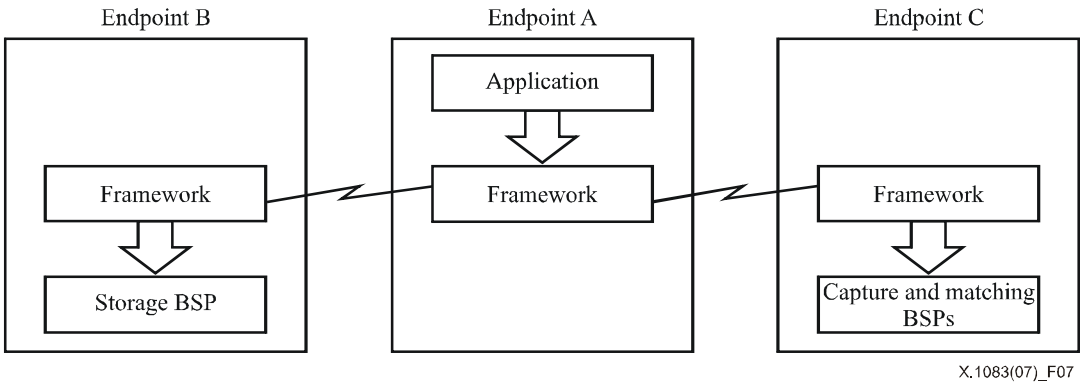

Figure 7 – An application communicating with two remote BSPs, one providing storage and the other providing capture and matching

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.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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**.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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	8.7.1	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

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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**.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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).

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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
        },
    ...
    }
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

```
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,
            ...
        }
}
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

```
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
    }
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

```
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
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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;

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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;
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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*.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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;
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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)
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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;
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

```
            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.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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)
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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),
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

```
            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;
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008

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;

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 24708:2008