INTERNATIONAL STANDARD

ISO/IEC 19784-1

First edition
2006-05-01

AMENDMENT 1
2007-12-01

Information technology — Biometric application programming interface

Part 1:
BioAPI specification

AMENDMENT 1: BioGUI specification

Technologies de l'information — Interface de programmation d'applications biométriques

Partie 1: Spécifications BioAPI

AMENDEMENT 1: Spécifications BioGUI

Reference number
ISO/IEC 19784-1:2006/Amd.1:2007(E)

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.

Amendment 1 to ISO/IEC 19784-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 37, *Biometrics*.

It defines a new version of BioAPI (BioAPI 2.1) that:

a) extends and improves the "application-controlled GUI" feature of BioAPI;

b) produces alignment between BioAPI and ISO/IEC 19785-1:2006, *Information technology — Common Biometric Exchange Formats Framework — Data element specification*

c) provides for other standards to modify BioAPI framework behaviour to support the use of biometric service providers (BSPs) that are remote from the controlling applications.

Introduction

With this amendment, ISO/IEC 19784-1 specifies two versions of BioAPI: 2.0 and 2.1. All the provisions that apply only to one version of BioAPI (either 2.0 or 2.1) are labeled as such.

The main difference of BioAPI 2.1 from BioAPI 2.0 is the support it provides for BioGUI. BioGUI stands for "BioAPI Graphical User Interface". The functionality specified in this amendment enables an application to control the display of graphics at enrollment, verification and identification as an alternative to using the graphical user interface provided by BSPs.

Secondly, BioAPI 2.1 also aligns the values of the type definition BioAPI_BIR_BIOMETRIC_TYPE with those specified in ISO/IEC 19785-1:2006.

Thirdly, it provides additional functions and parameters that are redundant in a purely local implementation of BIoAPI, but which enable other standards (interworking standards - see 4.29) to modify the behaviour of a BioAPI framework to support interactions between an application and remote BSPs.

Finally, it provides improvements to some of the functions and parameters defined for BioAPI 2.0, particularly in relation to support for tenprint capture, the electronic capture of ten human fingerprints.

This amendment redefines (in the specificaton of BioAPI 2.1) portions of the BioAPI 2.0 specification that define BioAPI types, macros, functions and callback functions (particularly those related to the application-controlled GUI feature) with a new set of definitions that provide more functionality. Some of the old BioAPI 2.0 definitions are completely replaced by new definitions (with the same or with different names), while others are extended by the addition of one or more parameters. Some types and functions are entirely new in BioAPI 2.1.

The resulting specification is expected to better meet the needs of biometric applications that wish to have full control of the user interface during enrollment, verification and identification, that need to be able to work with remote BSPs, or that need added functionality for interaction with local BSPs.

Information technology — Biometric application programming interface

Part 1: BioAPI specification

AMENDMENT 1: BioGUI specification

1) General amendment items

1-1) Replace the last paragraph of the Foreword with the following:

The first edition of this part of ISO/IEC 19784 was the first ISO/IEC standard on BioAPI. Previous versions of the specification were published by ANSI and the BioAPI Consortium. As the last official non-ISO specification was designated BioAPI 1.1, the versions specified in this part of ISO/IEC 19784 are designated BioAPI 2.0 and BioAPI 2.1.

1-2) Replace the first paragraph of the Introduction with:

This part of ISO/IEC 19784 provides a high-level generic biometric authentication model suited to most forms of biometric technology. No explicit support for multimodal biometrics is provided.

1-3) Insert the following paragraph after the third paragraph of the Introduction

This part of ISO/IEC 19784 specifies the behaviour of the BioAPI Framework when applications and BSPs are in the same system. Other interworking standards (see 4.29) specify modifications of that behaviour that enable both BSPs and Graphical User Interfaces to be remote from the system containing an application.

NOTE: ISO/IEC 24708 specifying the BioAPI Interworking Protocol (BIP) [6] is an example of an interworking standard.

2) Amendment items for the application-controlled GUI

2-1) Add the following text after 4.28:

4.29
interworking standards
standards that modify the behaviour of the BioAPI Framework (and BioAPI conformance requirements) to support the use of communications links to enable an application to interact with a remote BSP using a standardised protocol

4.30
test-verify
test-verification
one-to-one process of comparing a single biometric sample against a candidate of a biometric reference template at enrollment to determine whether the candidate template matches the test sample

4.31
enroll type
value denoting a pattern of suboperations performed by a BSP during an enroll operation

2-2) Add the following text after 6.1.8 d), after replacing "." at the end of c) with "; and":

e) providing graphical information to the application about an ongoing enrollment, verification, or identification operation.

2-3) Replace the heading of subclause 7.8 with the following text:

7.8 BioAPI_BIR_BIOMETRIC_TYPE (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-4) Add the following text after subclause 7.57 (BioAPI_VERSION):

7.58 BioAPI_BIR_BIOMETRIC_TYPE (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

A mask that describes the set of biometric types (factors) contained within a BioAPI BIR or supported by a BSP.

```
typedef uint32_t BioAPI_BIR_BIOMETRIC_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)
```

NOTE 1: BioAPI_TYPE_MULTIPLE_BIOMETRIC_TYPES is used to indicate that the biometric samples contained within the BDB (BIR BiometricData) include biometric samples from more than one type of biometric sensor unit (e.g., fingerprint and facial data). Location of the individual samples within the BDB is specified by the Format Owner and identified by the value of the Format Type.

NOTE 2: The condition NO VALUE AVAILABLE is indicated by setting the value to zero. BIRs that are not originally created by BioAPI BSPs should use this value when transformed into a BioAPI BIR if Biometric Type information is not available in the original source record. Transformed BIRs whose biometric type does not correspond to one of the defined types shall use the value for OTHER.

NOTE 3: The BioAPI BIR Biometric Type corresponds to the “CBEFF_BDB_biometric_type” in ISO/IEC 19785-1.

NOTE 4: Although “password” is not a biometric characteristic, BioAPI_TYPE_PASSWORD is included as a valid BioAPI_BIR_BIOMETRIC_TYPE to support its use a) for development & test environments, and b) as an additional authentication factor within a BioAPI application.

NOTE 5: The names of several biometric types are different in BioAPI 2.1 from the names used in BioAPI 2.0, with no change in semantics, and several new biometric types are present in BioAPI 2.1. Finally, the "thermal" types are present in BioAPI 2.0 but are absent from BioAPI 2.1. These differences are in order to align with ISO/IEC 19785-1.

NOTE 6: The names of the following values differ between BioAPI 2.0 and BioAPI 2.1, but their semantics is the same. The BioAPI 2.1 names are aligned with ISO/IEC 19785-1 (CBEFF).

Name in BioAPI 2.0	Name in BioAPI 2.1	Encoding
MULTIPLE	MULTIPLE_BIOMETRIC_TYPES	0x00000001
FACIAL_FEATURES	FACE	0x00000002
FINGERPRINT	FINGER	0x00000008
SIGNATURE_DYNAMICS	SIGNATURE_SIGN	0x00000080
KEYSTROKE_DYNAMICS	KEYSTROKE	0x00000100

NOTE 7: The following values are not present BioAPI 2.0 but are present in BioAPI 2.1 for consistency with ISO/IEC 19785-1 (CBEFF).

Added value	Encoding
NO_BIOTYPE_AVAILABLE	0x00000000
VEIN	0x00002000
DNA	0x00004000
EAR	0x00008000
FOOT	0x00010000
SCENT	0x00020000

2-5) Replace the heading of subclause 7.14 with the following text:

7.14 BioAPI_BIR_SUBTYPE (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-6) Add the following text after 7.58:

7.59 BioAPI_BIR_SUBTYPE (BioAPI 2.1)

7.59.1 This subclause applies only when the BioAPI version number in use is 2.1.

7.59.2 This identifies a subtype within the BDB type (specified in the BioAPI_BIR_BIOMETRIC_TYPE). Its values and their meaning are defined by the specifier of that BDB type.

NOTE: In the BioAPI 2.1 definition of this type, multiple bits can be set. This definition is aligned with ISO/IEC 19785-1.

```
typedef uint8_t BioAPI_BIR_SUBTYPE;

    #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)
```

7.59.3 This structure is a bitmask, with the bits defined as shown below. Bit 7 is used to indicate the interpretation of the lower-order bits. Bit positions zero (0) and one (1) always indicate left and right respectively. When bit position 7 is not set, these bit positions may apply to any biometric type; however, bit positions 2-6 are specific to the finger and vein biometric types. When bit position 7 is set, then the remaining bit positions apply to the vein biometric type only.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (Any)	Little	Ring	Middle	Pointer	Thumb	Right	Left
1 (Vein only)	Reserved	Reserved	Wrist	Back of Hand	Palm	Right	Left

NOTE: Vein biometric data may be obtained from the fingers (Bit 7 set to zero), or from the wrists, palm, or backs of either hand (Bit 7 set to 1).

7.59.4 Zero or more bits may be set.The abstract value NO VALUE AVAILABLE is indicated by setting all bits to zero.

NOTE The abstract value NO VALUE AVAILABLE can be used with BIRs that are not originally created by a BioAPI BSP but have been transformed from another data format. It can also be used when none of the other values are applicable or information is not available.

7.59.5 The bit positions BioAPI_BIR_SUBTYPE_THUMB through BioAPI_BIR_SUBTYPE_LITTLEFINGER can be used to identify the instance(s) of a biometric type related to the fingers.

NOTE: The BioAPI BIR Subtype corresponds to the “CBEFF_BDB_biometric_subtype” in ISO/IEC 19785-1.

7.59.6 If one or more of the bit positions BioAPI_BIR_SUBTYPE_THUMB through BioAPI_BIR_SUBTYPE_LITTLEFINGER are set, then either or both the bit positions BioAPI_BIR_SUBTYPE_LEFT_MASK and BioAPI_BIR_SUBTYPE_RIGHT_MASK shall also be set. When only the former is set, the subtype value designates one or more fingers of the left hand. When only the latter is set, the subtype value designates one or more fingers of the right hand. When both are set, the value designates one or more fingers of both hands (the same finger locations in both).

NOTE: It is not possible to designate, for example, a set of fingers consisting of the index finger of the left hand and the medium finger of the right hand.

7.59.7 When a value with multiple finger bits set occurs in a BIR header (or is used as an input parameter of a BioAPI function that performs capture), it is not strictly required that all the specified finger instances be actually present in the BDB of the BIR (or be actually captured). However, it is recommended that all the designated finger instances be present in the BDB except in the case of missing fingers and similar exceptional situations. Likewise, if the subtype field of a BIR designates multiple fingers, it is acceptable for the BDB of that BIR to include data about an extra finger if the subject happens to have six fingers in his or her hand.

7.59.8 If none of the finger bit positions BioAPI_BIR_SUBTYPE_THUMB through BioAPI_BIR_SUBTYPE_LITTLEFINGER are set, then the bit positions BioAPI_BIR_SUBTYPE_LEFT_MASK and BioAPI_BIR_SUBTYPE_RIGHT_MASK can be used to identify an instance of a biometric type for which there is one left instance and one right instance (for example, iris), or also the single instance of a biometric type for which there is only one instance (for example, face).

7.59.9 Vein data taken from one or more fingers is indicated by setting bit position 7 to zero, setting one or both of bit positions 0 and 1, and setting one or more of the finger bit positions (3-7). Vein data taken from one of the other sources (wrist, palm, or back of hand) is indicated by setting bit position 7 to one, setting one or both of bit positions 0 and 1, and setting one of bit positions 3-5 as appropriate).

NOTE 1: BIRs that are not originally created by a BioAPI BSP but have been transformed from another data format and for which subtype information is not available may use the NO VALUE AVAILABLE condition.

NOTE 2: The BioAPI BIR Subtype corresponds to the “CBEFF_BDB_biometric_subtype” in ISO/IEC 19785-1.

NOTE 3: This structure is primarily used within a BIR header; however, it is also used as an input parameter for functions that capture biometric data. The BioAPI_NO_SUBTYPE_AVAILABLE value is used in the BIR header when subtype information is not available. BioAPI_NO_SUBTYPE_AVAILABLE is also used as a function parameter when the application allows the BSP to determine which subtype is to be captured.

7.59.10 The use of the bit position BioAPI_BIR_SUBTYPE_MULTIPLE is not recommended when the BioAPI version number in use is 2.1.

2-7) Replace subclause 7.57 (BioAPI_VERSION) with the following text:

7.57 BioAPI_VERSION

This type is used to represent the version of the BioAPI or FPI specification to which components or data have been implemented. This type is used in the function BioAPI_Init, within the BIR header, and within schemas in the component registry.

The two following values are specified in this edition of this International Standard:

(1) 32 decimal (20 hex), corresponding to a Major value of 2 and a Minor value of 0, and representing BioAPI 2.0; and

(2) 33 decimal (21 hex), corresponding to a Major value of 2 and a Minor value of 1, and representing BioAPI 2.1.

```
typedef uint8_t BioAPI_VERSION;
```

NOTE 1: This type is not used for product versions, which are generally represented by strings.

NOTE 2: The BioAPI Version used within the BIR header corresponds to the “CBEFF_patron_header_version” in ISO/IEC 19785-1.

The BioAPI Version is a concatenation of the major and minor version values such that first hex digit represents the major version and the second hex digit represents the minor version:

BioAPI_VERSION 0xnm

where n = MajorVersion and m=MinorVersion:

2-8) Replace the heading of subclause 7.16 (BioAPI_BSP_SCHEMA) with the following text:

7.16 BioAPI_BSP_SCHEMA (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-9) Add the following text after 7.59:

7.60 BioAPI_BSP_SCHEMA (BioAPI 2.1)

7.60.1 This subclause applies only when the BioAPI version number in use is 2.1.

7.60.2 Information about a BSP, maintained in the component registry.

```
typedef struct bioapi_bsp_schema {
        BioAPI_UUID BSPUuid;
        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;
        uint32_t  MaxNumEnrollInstances;
        uint8_t   *HostingEndpointIRI;
        BioAPI_UUID  BSPAccessUUID;
   }BioAPI_BSP_SCHEMA;
```

7.60.3 Definitions

BSPUuid – UUID of the BSP.

BSPDescription – A NUL-terminated string containing a text description of the BSP.

Path – A pointer to a NUL-terminated string containing the path of the file containing the BSP executable code, including the filename. The path may be a URL. This string shall consist of ISO/IEC 10646 characters encoded in UTF-8 (see ISO/IEC 10646, Annex D).

NOTE: When BioAPI_BSP_SCHEMA is used within a function call, the component that receives the call allocates the memory for the *Path* schema element and the calling component frees the memory.

SpecVersion – Major/minor version number of the BioAPI specification to which the BSP was implemented.

ProductVersion – The version string of the BSP software.

Vendor – A NUL-terminated string containing the name of the BSP vendor.

BSPSupportedFormats – A pointer to an array of BioAPI_BIR_BIOMETRIC_DATA_FORMAT structures specifying the supported BDB formats.

NumSupportedFormats – Number of supported formats contained in BSPSupportedFormats.
FactorsMask – A mask which indicates which biometric types are supported by the BSP.
Operations – A mask which indicates which operations are supported by the BSP.

Options – A mask which indicates which options are supported by the BSP.

PayloadPolicy – Threshold setting (maximum FMR value) used to determine when to release the payload after successful verification.

MaxPayloadSize – Maximum payload size (in bytes) that the BSP can accept.

DefaultVerifyTimeout – Default timeout value in milliseconds used by the BSP for ***BioAPI_Verify*** operations when no timeout is specified by the application.

DefaultIdentifyTimeout – Default timeout value in milliseconds used by the BSP for ***BioAPI_Identify*** and ***BioAPI_IdentifyMatch*** operations when no timeout is specified by the application.

DefaultCaptureTimeout – Default timeout value in milliseconds used by the BSP for ***BioAPI_Capture*** operations when no timeout is specified by the application.

DefaultEnrollTimeout – Default timeout value in milliseconds used by the BSP for ***BioAPI_Enroll*** operations when no timeout is specified by the application.

DefaultCalibrateTimeout **–** Default timeout value in milliseconds used by the BSP for sensor calibration operations when no timeout is specified by the application.

MaxBSPDbSize – Maximum size of a BSP-controlled BIR database.

NOTE 1: Applies only when a BSP is only capable of directly managing a single archive unit.
NOTE 2: A value of zero means that no information about the database size is being provided for one of the following three reasons:

a) databases are not supported,

b) it is capable of managing multiple units (either directly or through a BFP interface), each of which may have a different “maximum size” and information about these units will be provided as part of the insert notification (part of Unit Schema), or

c) one archive unit is supported, but the information is not given here – it will be provided in the insert notification.

MaxIdentify – Largest population supported by the identify function. Unlimited = FFFFFFFF.

MaxNumEnrollInstances – The maximum number of distinct instances that a BSP can create reference templates for in one enroll operation. This information can be useful to an application that uses the application-controlled GUI feature.

HostingEndpointIRI – An IRI identifying the framework whose component registry contains a registration of the BSP. This parameter shall be ignored by frameworks conforming to this part of this International Standard and shall be set to NULL by an application. It is provided to support interworking standards, which may specify the use of identical BSPs present on multiple computers from within an application running on the same or a different computer.

BSPAccessUUID – A UUID, unique within the scope of an application, which the application may use to refer to the BSP as an alternative to the BSP product UUID. This parameter shall be ignored by frameworks conforming to this part of this International Standard and can be set to any UUID value by an application. It is provided to support interworking standards, which may specify the use of identical BSPs present on multiple computers from within an application running on the same or a different computer.

NOTE: The "BSPAccess_UUID" and the "HostingendpointIRI" are part of the definition of the C type **BioAPI_BSP_SCHEMA**, but are not part of the BSP schema information stored in the component registry (see Table 3).

7.60.4 See subclause 10.1.2 and 10.2.1 for further explanation of schema elements and for BSP insertion of information into the component registry.

2-10) Replace the parameter definitions in 7.28.2 with the following text:

BSPUuid (input) – The UUID of the BSP raising the event.

UnitID (input) – The unit ID of the BioAPI Unit associated with the event.

AppNotifyCallbackCtx (input) – A generic pointer to context information that was provided in the call to **BioAPI_BSPLoad** that established the event handler.

UnitSchema (input) – A pointer to the unit schema of the BioAPI Unit associated with the event.

EventType (input) – The **BioAPI_EVENT** that has occurred.

2-11) Replace the heading of subclause 7.31 (BioAPI_GUI_BITMAP) with the following text:

7.31 BioAPI_GUI_BITMAP (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-12) Add the following text after 7.60:

7.61 BioAPI_GUI_BITMAP (BioAPI 2.1)

7.61.1 This subclause applies only when the BioAPI version number in use is 2.1.

7.61.2 This structure provides a graphic for display by the application.

```
typedef struct bioapi_gui_bitmap {
   BioAPI_BIR_SUBTYPE_MASK SubtypeMask;
      /* The subtype (or subtypes) of the sample represented
      by the bitmap */
   uint32_t  Width;
      /* Width of bitmap in pixels (number of pixels for each line) */
   uint32_t  Height;
      /* Height of bitmap in pixels (number of lines) */
   BioAPI_DATA  Bitmap;
} BioAPI_GUI_BITMAP;
```

7.61.3 Definitions

`Bitmap` – A series of 8-bit grayscale pixels (where ‘00’ = black and ‘FF’ = white), read from left to right, top to bottom, as specified in Table Amd.1-1.

Table Amd.1-1 — GUI bitmap format

Byte position	Meaning	Description
0	line 0, pixel 0	first pixel of first line
1	line 0, pixel 1	second pixel of first line
…	…	
`Width` -1	line 0, pixel (`Width` -1)	last pixel of first line
`Width`	line 1, pixel 0	first pixel of second line
…	…	
(`Width` * `Height`) - 1	line (`Width` - 1), pixel (`Height` - 1)	last line, last pixel

NOTE: This is used with the application-controlled GUI option. See `BioAPI_GUI_STATE_EVENT_HANDLER` and `BioAPI_GUI_PROGRESS_EVENT_HANDLER` descriptions in 7.71.

2-13) Replace the heading of subclause 7.32 (BioAPI_GUI_MESSAGE) with the following text:

7.32 BioAPI_GUI_MESSAGE (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-14) Add the following text after 7.61:

7.62 BioAPI_GUI_ENROLL_TYPE (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

The type BioAPI_GUI_ENROLL_TYPE indicates the enroll type of a BSP, and is defined as follows:

```
typedef uint32_t BioAPI_GUI_ENROLL_TYPE;

#define BioAPI_GUI ENROLL_TYPE TEST_VERIFY      (0x00000001)

#define BioAPI_GUI ENROLL_TYPE MULTIPLE_CAPTURE (0x00000002)
```

The enroll type of a BSP denotes a pattern of suboperations performed by the BSP during an enroll operation.

The bit position BioAPI_GUI ENROLL_TYPE TEST_VERIFY indicates, when set, that the BSP supports test verification at enrollment. During an enroll operation, an application receives GUI state event notification callbacks for two capture suboperations, GUI state event notification callbacks for a process suboperation, GUI state event notification callbacks for a create template suboperation, and GUI state event notification callbacks for a verify match suboperation;

The bit position BioAPI_GUI ENROLL_TYPE MULTIPLE_CAPTURE indicates, when set, that the BSP supports multiple capture suboperations at enrollment. During an enroll operation, an application receives GUI state event notification callbacks for multiple capture suboperations followed by GUI state event notification callbacks for a create template suboperation.

The two bit positions shall not be set at the same time.

NOTE: This restriction is imposed because an application, in order to support a combination of Test-Verify and Multiple-Capture, would need more information about the timing of the suboperations in order to decide the layout of its screen during enrollment.

7.63 BioAPI_GUI_BITMAP_ARRAY (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

The type **BioAPI_GUI_BITMAP_ARRAY** is defined as follows:

```
typedef struct bioapi_gui_bitmap_array {
   uint32_t         NumberOfMembers;
   BioAPI_GUI_BITMAP    *Bitmaps;
      /* A pointer to an array of BioAPI_GUI_BITMAPs */
} BioAPI_GUI_BITMAP_ARRAY;
```

7.64 BioAPI_BIR_SUBTYPE_MASK (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

The type **BioAPI_BIR_SUBTYPE_MASK** is defined as follows:

```
typedef uint32_t BioAPI_BIR_SUBTYPE_MASK;

#define BioAPI_BIR_SUBTYPE_LEFT_BIT                (0x0001)
#define BioAPI_BIR_SUBTYPE_RIGHT_BIT               (0x0002)
#define BioAPI_BIR_SUBTYPE_LEFT_THUMB_BIT          (0x0004)
#define BioAPI_BIR_SUBTYPE_LEFT_POINTERFINGER_BIT  (0x0008)
#define BioAPI_BIR_SUBTYPE_LEFT_MIDDLEFINGER_BIT   (0x0010)
#define BioAPI_BIR_SUBTYPE_LEFT_RINGFINGER_BIT     (0x0020)
#define BioAPI_BIR_SUBTYPE_LEFT_LITTLEFINGER_BIT   (0x0040)
#define BioAPI_BIR_SUBTYPE_RIGHT_THUMB_BIT         (0x0080)
#define BioAPI_BIR_SUBTYPE_RIGHT_POINTERFINGER_BIT    (0x0100)
#define BioAPI_BIR_SUBTYPE_RIGHT_MIDDLEFINGER_BIT     (0x0200)
#define BioAPI_BIR_SUBTYPE_RIGHT_RINGFINGER_BIT       (0x0400)
#define BioAPI_BIR_SUBTYPE_RIGHT_LITTLEFINGER_BIT     (0x0800)
#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)
```

NOTE 1: The VEIN values are not present in BioAPI 2.0.

The bit positions **BioAPI_BIR_SUBTYPE_LEFT_BIT** and **BioAPI_BIR_SUBTYPE_RIGHT_BIT** can be used to identify the instance of a biometric modality for which there is one "left" instance and one "right" instance.

NOTE 2: Iris and hand geometry are examples of such modalities.

The other bit positions (from **BioAPI_BIR_SUBTYPE_LEFT_THUMB_BIT** through **BioAPI_BIR_SUBTYPE_RIGHT_LITTLEFINGER_BIT**) can be used to identify the instance of a biometric modality related to the fingers.

NOTE 3: Fingerprint is one such modality.

The value zero (no bit positions set) can be used with biometric modalities for which there is only one instance.

NOTE 4: Signature/sign is one such modality.

7.65 BioAPI_GUI_EVENT_SUBSCRIPTION (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.
This type carries information about an existing named GUI event subscription. It identifies the subscriber application and the named subscription, and indicates which types of GUI events are in the scope of the subscription. This type is specified for use in the **BioAPI_QueryGUIEventSubscriptions** function.

A named GUI event subscription is one that was created by a call to **BioAPI_SubscribeToGUIEvents** specifying a non-**NULL** GUI event subscription UUID. The framework calls the event handlers specified in a named subscription to notify BSP-generated GUI events that have been redirected to that named subscription (see 8.3.7), and to notify application-generated GUI events (see 8.3.3, 8.3.4, and 8.3.5) directed to that named subscription.

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

SubscriberEndpointIRI – An IRI -see RFC3987- (originally provided by the framework) that identifies the application that created the named GUI event subscription. This shall be **NULL** if the subscriber application is the same as the current application.

GUIEventSubscriptionUuid – A UUID (originally provided by the subscriber application) that identifies the named GUI event subscription.

GUISelectEventSubscribed – Indicates whether GUI select events are in the scope of the subscription (a non-**NULL** callback address was originally provided by the subscriber application for the GUI select event handler).

GUIStateEventSubscribed – Indicates whether GUI state events are in the scope of the subscription (a non-**NULL** callback address was originally provided by the subscriber application for the GUI state event handler).

GUIProgressEventSubscribed – Indicates whether GUI progress events are in the scope of the subscription (a non-**NULL** callback address was originally provided by the subscriber application for the GUI progress event handler).

7.66 BioAPI_GUI_MOMENT (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

An enumeration of the moments in which:

a) a GUI select event can be generated with respect to a suboperation cycle, or

b) a GUI state or progress event can be generated with respect to a suboperation.

```
typedef uint32_t BioAPI_GUI_MOMENT;

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

The values **BioAPI_GUI_MOMENT_BEFORE_START** and **BioAPI_GUI_MOMENT_AFTER_END** can occur in all types of GUI event notifications, whereas the value **BioAPI_GUI_MOMENT_DURING** can only occur in GUI progress event notifications.

When the value **BioAPI_GUI_MOMENT_BEFORE_START** occurs in a GUI select event notification, it indicates that the BSP is ready to start a new suboperation cycle. A suboperation cycle consists of a single execution of the sequence of suboperations that comprise an operation (see 7.68). There is no limit to the number of suboperation cycles (successful or not) that the application can request to be performed for an operation. However, all the data produced in each cycle shall be discarded when a subsequent cycle is started, and therefore the outcome of an operation (including the return value and the data produced) is always the outcome of its last (or only) suboperation cycle (except in the case of cancellation).

When the value **BioAPI_GUI_MOMENT_AFTER_END** occurs in a GUI select event notification callback, it indicates that the current suboperation cycle has ended. This value does not indicate whether the suboperation cycle has been successful or not, as that information is provided by a separate parameter of the GUI select event notification that contains the result value of the suboperation cycle.

When the value **BioAPI_GUI_MOMENT_BEFORE_START** occurs in a GUI state event notification callback, it indicates that the BSP is ready to start a new suboperation within the current suboperation cycle.

When the value **BioAPI_GUI_MOMENT_AFTER_END** occurs in a GUI state event notification callback, it indicates that the current suboperation performmed by the BSP has ended. This value does not indicate whether the suboperation has been successful or not, as that information is provided by a separate parameter of the GUI state event notification which carries the result value of the suboperation cycle.

When the value **BioAPI_GUI_MOMENT_BEFORE_START** occurs in a GUI progress event notification callback, it indicates that the BSP is ready to start a new suboperation within the current suboperation cycle. A GUI progress event with this moment value may be redundant when a GUI state event is also produced, and may be omitted. However, such a GUI progress event is useful for those operations (such as the identify match operation) that do not produce GUI state events.

When the value **BioAPI_GUI_MOMENT_DURING** occurs in a GUI progress event notification callback, it indicates that the suboperation is in progress.

When the value **BioAPI_GUI_MOMENT_AFTER_END** occurs in a GUI progress event notification callback, it indicates that the current suboperation performmed by the BSP has ended. A GUI progress event with this moment value may be redundant when a GUI state event is also produced, and may be omitted. However, such a GUI progress event is useful for those operations (such as the identify match operation) that do not produce GUI state events.

NOTE: This type is used with the application-controlled GUI feature (see 7.71).

2-15) Replace the heading of subclause 7.33 (BioAPI_GUI_PROGRESS) with the following text:

7.33 BioAPI_GUI_PROGRESS (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-16) Add the following text after 7.66:

7.67 BioAPI_GUI_PROGRESS (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

A value in the range 0 to 100 indicating the percent of completion of a suboperation, intended for display to the subject or an operator.

```
typedef uint8_t BioAPI_GUI_PROGRESS;
```

NOTE: This is used with the application-controlled GUI option. See **BioAPI_GUI_PROGRESS_EVENT** description in 7.71.

7.68 BioAPI_GUI_OPERATION (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

An enumeration of all BioSPI functions (operations) during which a BSP can generate GUI events.

```
typedef uint8_t BioAPI_GUI_OPERATION;

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

The values of this type occur in GUI event notifications, and identify the type of BioSPI function that was in execution when the BSP produced the GUI event, and to which the GUI event refers.

Table Amd.1-2 specifies the correspondence between BioSPI functions and operations.

Table Amd.1-2 — BioSPI functions and operations

BioSPI function	`BioAPI_GUI_OPERATION` value	Common name of the operation
`BioSPI_Capture`	`BioAPI_GUI_OPERATION_CAPTURE`	capture operation
`BioSPI_Process`	`BioAPI_GUI_OPERATION_PROCESS`	process operation
`BioSPI_CreateTemplate`	`BioAPI_GUI_OPERATION_CREATETEMPLATE`	create template operation
`BioSPI_VerifyMatch`	`BioAPI_GUI_OPERATION_VERIFYMATCH`	verify match operation
`BioSPI_IdentifyMatch`	`BioAPI_GUI_OPERATION_IDENTIFYMATCH`	identify match operation
`BioSPI_Verify`	`BioAPI_GUI_OPERATION_VERIFY`	verify operation
`BioSPI_Identify`	`BioAPI_GUI_OPERATION_IDENTIFY`	identify operation
`BioSPI_Enroll`	`BioAPI_GUI_OPERATION_ENROLL`	enroll operation

Table Amd.1-3 specifies which GUI events can be generated by a BSP during the execution of each type of operation.

Table Amd.1-3 — GUI events generated in each operation

Operation	GUI select events	GUI state events	GUI progress events
capture operation	X	X	X
process operation			X
Create template operation			X
verify match operation			X
identify match operation			X
verify operation	X	X	X
identify operation	X	X	X
enroll operation	X	X	X

2-17) Replace the heading of subclause 7.34 (BioAPI_GUI_RESPONSE) with the following text:

7.34 BioAPI_GUI_RESPONSE (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-18) Add the following text after 7.68:

7.69 BioAPI_GUI_RESPONSE (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

An enumeration of the possible actions to be performed by a BSP after a GUI event notification callback made by the BSP has returned control to the BSP. Before returning from a notification callback, an application assigns a value of this type to an output parameter of the callback (**`Response`**).

```
typedef uint8_t BioAPI_GUI_RESPONSE;

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

The value **`BioAPI_GUI_RESPONSE_OP_COMPLETE`** can only be returned in response to a GUI select event notification callback with the moment value **`BioAPI_GUI_MOMENT_AFTER_END`**. It indicates that the BSP should consider the entire operation complete and should cause the original function call to return the result code of the current suboperation cycle (which can be either **`BioAPI_OK`** or an error code). After this response from the application, no further GUI event notification callbacks are expected for the same operation.

The value **`BioAPI_GUI_RESPONSE_OP_CANCEL`** can be returned in response to any GUI event notification callback. It indicates that the BSP should abort the entire operation and cause the original function call to return an error code. After this response from the application, no further GUI event notification callbacks are expected from the BSP for the same operation.

The value **`BioAPI_GUI_RESPONSE_CYCLE_START`** can only be returned in response to a GUI select event notification callback with the moment value **`BioAPI_GUI_MOMENT_BEFORE_START`**. It indicates that the BSP should start the suboperation cycle.

The value **`BioAPI_GUI_RESPONSE_CYCLE_RESTART`** can only be returned in response to a GUI select event notification callback with the moment value **`BioAPI_GUI_MOMENT_AFTER_END`** or in response to a GUI state or progress event notification callback with any moment value. It indicates that the BSP should discard all the information produced during the current suboperation cycle and should start a new suboperation cycle. After this response from the application, a GUI select event notification callback with the moment value **`BioAPI_GUI_MOMENT_BEFORE_START`** is expected from the BSP.

The value **`BioAPI_GUI_RESPONSE_SUBOP_START`** can only be returned in response to a GUI state event notification callback with the moment value **`BioAPI_GUI_MOMENT_BEFORE_START`**. It indicates that the BSP should start the suboperation.

The value **`BioAPI_GUI_RESPONSE_SUBOP_NEXT`** can only be returned in response to a GUI state event notification callback with the moment value **`BioAPI_GUI_MOMENT_AFTER_END`**. It indicates that the BSP should either perform the next suboperation in the current suboperation cycle (if there are more suboperations to be performed) or exit the suboperation cycle (if all the suboperations in the cycle have been performed). After this response from the application, either a GUI state event notification callback with the moment value **`BioAPI_GUI_MOMENT_BEFORE_START`** or a GUI select event notification callback with the moment value **`BioAPI_GUI_MOMENT_AFTER_END`** is expected from the BSP (respectively).

The value **`BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE`** can only be returned in response to a GUI progress event notification callback. It indicates that the BSP should continue performing the suboperation.

The value **`BioAPI_GUI_RESPONSE_PROGRESS_ABORT`** can only be returned in response to a GUI progress event notification callback. It indicates that the BSP should abort the suboperation and produce an error code. After this response from the application, a GUI state event notification callback with the moment value **`BioAPI_GUI_MOMENT_AFTER_END`** is expected from the BSP.

The value **`BioAPI_GUI_RESPONSE_RECAPTURE`** can only be returned in response to a GUI state event notification callback with the moment value **`BioAPI_GUI_MOMENT_AFTER_END`**. It indicates that the BSP should discard one of the samples captured in the current suboperation cycle (possibly also discarding any reference template and any processed sample created from the sample being discarded) and perform another capture suboperation to produce a new sample. After this response from the application, a GUI state event notification callback with the moment value **`BioAPI_GUI_MOMENT_BEFORE_START`** (for a capture suboperation with the purpose of enrollment) is expected from the BSP. When the capture suboperation is completed, the BSP shall perform any other suboperations required to complete the current suboperation cycle.

The value **`BioAPI_GUI_RESPONSE_DEFAULT`** shall be interpreted as the one among the following values which applies to the specific situation in which it is returned: **`BioAPI_GUI_RESPONSE_CYCLE_START`**, **`BioAPI_GUI_RESPONSE_SUBOP_START`**, **`BioAPI_GUI_RESPONSE_PROCESS_CONTINUE`**, **`BioAPI_GUI_RESPONSE_SUBOP_NEXT`**, or **`BioAPI_GUI_RESPONSE_OP_COMPLETE`**.

If an application returns a response value other than those permitted in each situation, the BSP shall abort the entire operation and cause the original function call to return an error code. No further GUI event notification callbacks are expected from the BSP for the same operation.

Table Amd.1-4 summarizes the compatibility between types of GUI events, moment values, and responses.

Table Amd.1-4 — GUI responses and GUI events

Response from the application	GUI select event (before start)	GUI state event (before start)	GUI progress event (before start)	GUI progress event (during)	GUI progress event (after end)	GUI state event (after end)	GUI select event (after end)
`BioAPI_GUI_RESPONSE_DEFAULT`	X	X	X	X	X	X	X
`BioAPI_GUI_RESPONSE_CYCLE_START`	X						
`BioAPI_GUI_RESPONSE_SUBOP_START`		X					
`BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE`			X	X	X		
`BioAPI_GUI_RESPONSE_SUBOP_NEXT`						X	
`BioAPI_GUI_RESPONSE_OP_COMPLETE`							X
`BioAPI_GUI_RESPONSE_PROGRESS_ABORT`			X	X	X		
`BioAPI_GUI_RESPONSE_RECAPTURE`						X	
`BioAPI_GUI_RESPONSE_CYCLE_RESTART`		X	X	X	X	X	X
`BioAPI_GUI_RESPONSE_OP_CANCEL`	X	X	X	X	X	X	X

7.70 BioAPI_GUI_SUBOPERATION (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

An enumeration of the possible types of suboperations performed by a BSP during an operation, to be reported to the application in GUI event notifications.

```
typedef uint8_t BioAPI_GUI_SUBOPERATION;
```

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

The values of this type occur in GUI event notifications and identify a type of suboperation of the current operation for which the GUI event has been generated.

Table Amd.1-5 shows the types of suboperations (and their number) performed by a BSP during each type of operation (in order). For the operations **BioAPI_GUI_OPERATION_CAPTURE**, **BioAPI_GUI_OPERATION_VERIFY**, **BioAPI_GUI_OPERATION_IDENTIFY**, and **BioAPI_GUI_OPERATION_ENROLL**, the number of suboperations is relative to a complete suboperation cycle (see 7.66) and does not take account of possible recaptures done within the cycle (see 7.69).

Table Amd.1-5 — Suboperations performed in each operation

Operation	Suboperations
BioAPI_GUI_OPERATION_CAPTURE with the purpose of verification or identification	**BioAPI_GUI_SUBOPERATION_CAPTURE** (one) **BioAPI_GUI_SUBOPERATION_PROCESS** (zero or one)
BioAPI_GUI_OPERATION_CAPTURE with the purpose of enrollment (when the enroll type is Test-Verify)	**BioAPI_GUI_SUBOPERATION_CAPTURE** (two) **BioAPI_GUI_SUBOPERATION_CREATETEMPLATE** (one) **BioAPI_GUI_SUBOPERATION_PROCESS** (one) **BioAPI_GUI_SUBOPERATION_VERIFYMATCH** (one)
BioAPI_GUI_OPERATION_CAPTURE with the purpose of enrollment (when the enroll type is Multiple-Capture)	**BioAPI_GUI_SUBOPERATION_CAPTURE** (multiple) **BioAPI_GUI_SUBOPERATION_CREATETEMPLATE** (one)
BioAPI_GUI_OPERATION_PROCESS	**BioAPI_GUI_SUBOPERATION_PROCESS** (one)
BioAPI_GUI_OPERATION_CREATETEMPLATE	**BioAPI_GUI_SUBOPERATION_CREATETEMPLATE** (one)
BioAPI_GUI_OPERATION_VERIFYMATCH	**BioAPI_GUI_SUBOPERATION_VERIFYMATCH** (one)
BioAPI_GUI_OPERATION_IDENTIFYMATCH	**BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH** (one)
BioAPI_GUI_OPERATION_VERIFY	**BioAPI_GUI_SUBOPERATION_CAPTURE** (one) **BioAPI_GUI_SUBOPERATION_PROCESS** (one) **BioAPI_GUI_SUBOPERATION_VERIFYMATCH** (one)
BioAPI_GUI_OPERATION_IDENTIFY	**BioAPI_GUI_SUBOPERATION_CAPTURE** (one) **BioAPI_GUI_SUBOPERATION_PROCESS** (one) **BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH** (one)
BioAPI_GUI_OPERATION_ENROLL (when the enroll type is Test-Verify)	**BioAPI_GUI_SUBOPERATION_CAPTURE** (two) **BioAPI_GUI_SUBOPERATION_CREATETEMPLATE** (one) **BioAPI_GUI_SUBOPERATION_PROCESS** (one) **BioAPI_GUI_SUBOPERATION_VERIFYMATCH** (one)
BioAPI_GUI_OPERATION_ENROLL (when the enroll type is Multiple-Capture)	**BioAPI_GUI_SUBOPERATION_CAPTURE** (multiple) **BioAPI_GUI_SUBOPERATION_CREATETEMPLATE** (one)

2-19) Replace the heading of subclause 7.35 (BioAPI_GUI_STATE) with the following text:

7.35 BioAPI_GUI_STATE (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-20) Replace the heading of subclause 7.36 (BioAPI_GUI_STATE_CALLBACK) with the following text:

7.36 BioAPI_GUI_STATE_CALLBACK (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-21) Replace the heading of subclause 7.37 (BioAPI_GUI_STREAMING_CALLBACK) with the following text:

7.37 BioAPI_GUI_STREAMING_CALLBACK (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-22) Add the following text after 7.70:

7.71 GUI Events

This subclause applies only when the BioAPI version number in use is 2.1.

7.71.1 BioAPI_GUI_SELECT_EVENT_HANDLER (BioAPI 2.1)

7.71.1.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioAPI_GUI_SELECT_EVENT_HANDLER)
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    BioAPI_GUI_ENROLL_TYPE EnrollType,
    const void *GUISelectEventHandlerCtx,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
    int32_t MaxNumEnrollSamples,
    BioAPI_BIR_SUBTYPE_MASK SelectableInstances,
    BioAPI_BIR_SUBTYPE_MASK *SelectedInstances,
    BioAPI_BIR_SUBTYPE_MASK CapturedInstances,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response);
```

7.71.1.2 Description

This is a function pointer type for an application's GUI event handler function that is to handle GUI select event notification callbacks coming from a BSP through the framework. In order to receive GUI select event notifications, an application shall register a callback function of type **`BioAPI_GUI_SELECT_EVENT_HANDLER`** by providing the callback address of the function, along with a context address, in a call to **`BioAPI_SubscribeToGUIEvents`** (see 8.3.8).

The framework makes a callback to an application function of this type each time it receives an incoming callback to a function of type **`BioSPI_GUI_SELECT_EVENT_HANDLER`** which it exposes to a BSP. The parameters of the callback (except **`GUISelectEventHandlerCtx`**) shall be set from the parameters of the incoming callback with the same names; the parameter **`GUISelectEventHandlerCtx`** shall be set from the GUI select context address originally provided by the application in its call to **`BioAPI_SubscribeToGUIEvents`**; and the callback address shall be set from the GUI select callback address originally provided by the application in the call to **`BioAPI_SubscribeToGUIEvents`**.

A BSP can generate GUI select events only during the execution of a call to **`BioAPI_Capture`**, **`BioAPI_Verify`**, **`BioAPI_Identify`**, or **`BioAPI_Enroll`**. A GUI select event with the moment value **`BioAPI_GUI_MOMENT_BEFORE_START`** is generated before the start of each suboperation cycle, and a GUI select event with the moment value **`BioAPI_GUI_MOMENT_AFTER_END`** is generated after the end of each suboperation cycle (see 7.66).

The main purpose of the GUI select event notification generated before the start of a suboperation cycle is to inform the application that a new suboperation cycle is ready to start and to allow the application to select the instance(s) to be captured (where such selection is supported by the BSP and is consistent with the biometric modality in use). The main purpose of the GUI select event notification generated after the end of a suboperation cycle is to inform the application about the outcome of the suboperation cycle. The application shall respond to each GUI select notification by indicating whether the suboperation cycle should start, the operation should be considered completed, a new suboperation cycle should be started, or the operation should be canceled (see 7.69).

If the application specifies one or more instances in the **`Subtype`** parameter of the original function call (**`BioAPI_Capture`**, **`BioAPI_Enroll`**, **`BioAPI_Verify`**, or **`BioAPI_Identify`**) and this parameter is supported by the BSP, then the parameter **`SelectableInstances`** is set based on the value of the **`Subtype`** parameter.

The GUI select event handler function and any functions invoked (directly or indirectly) by that function shall not call any BioAPI function.

If the application returns a value different from **`BioAPI_OK`**, the original function will immediately return to the caller with an error code.

7.71.1.3 Parameters

`BSPUuid` (input) – A UUID identifying the BSP to which the GUI event is related.

`UnitID` (input) – The unit ID of the sensor unit of the BSP to which the GUI event is related.

`BSPHandle` (input) – The handle of the BSP attach session to which the GUI event is related.

`EnrollType` (input) – The enroll type of the BSP. This parameter is only meaningful if *Operation* is BioAPI_GUI_OPERATION_ENROLL.

`GUISelectEventHandlerCtx` (input) – The context address orginally provided by the subscriber application as part of the GUI event subscription.

`Operation` (input) – A value indicating the type of operation for which the GUI event has been generated. Only the following values are permitted:

— **BioAPI_GUI_OPERATION_CAPTURE**

— **BioAPI_GUI_OPERATION_VERIFY**

— **BioAPI_GUI_OPERATION_IDENTIFY**

— **BioAPI_GUI_OPERATION_ENROLL**

Moment (input) – A value indicating whether the GUI event has been generated before the start of the suboperation cycle or after the end of the suboperation cycle. Only the following values are permitted:

— **BioAPI_GUI_MOMENT_BEFORE_START**

— **BioAPI_GUI_MOMENT_AFTER_END**

ResultCode (input) – This parameter is significant only if **Moment** is **BioAPI_GUI_MOMENT_AFTER_END**, and shall be set to zero otherwise. When significant, it shall contain the result code of the suboperation cycle that has just ended. The result code can be either **BioAPI_OK** or a BioAPI error code.

MaxNumEnrollSamples (input) – This parameter is significant only if **Moment** is **BioAPI_GUI_MOMENT_BEFORE_START**. When significant, it shall contain the maximum number of samples that the BSP will be able to capture in the suboperation cycle that is about to start.

SelectableInstances (input) – This parameter is significant only if **Moment** is **BioAPI_GUI_MOMENT_BEFORE_START**. When significant, it shall contain a bitmask that specifies which instance(s) of the biometric modality can be selected by the application for capturing in the suboperation cycle that is about to start. Some BSPs support the selection of multiple instances of a modality because they are able to create, in a single capture suboperation, a single BIR containing data pertaining to multiple instances.

SelectedInstances (output) – This output parameter is significant only if **Moment** is **BioAPI_GUI_MOMENT_BEFORE_START**. When significant, it shall contain a bitmask that specifies which instance(s) of the biometric modality have been selected by the application for capturing in the suboperation cycle that is about to start.

CapturedInstances (input) – This parameter is significant only if **Moment** is **BioAPI_GUI_MOMENT_AFTER_END**, and shall be set to zero otherwise. When significant, it shall contain a bitmask that specifies which instance(s) have been successfully captured by the BSP within the suboperation cycle that has just ended.

Text (input) – A textual message, consisting of a string of ISO/IEC 10646 characters encoded in UTF-8, which is intended for display to the subject or to an operator. The content of the message, as well as the language in which it is expressed, are not standardized.

Response (output) – A value indicating the response from the application to the GUI select event notification (see 7.69).

NOTE: See also C.7.

7.71.1.4 Return Value

A **BioAPI_RETURN** value indicating success or specifying a particular error condition. The value **BioAPI_OK** indicates success. All other values represent an error condition.

7.71.1.5 Errors

BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED

7.71.2 BioAPI_GUI_STATE_EVENT_HANDLER (BioAPI 2.1)

7.71.2.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioAPI_GUI_STATE_EVENT_HANDLER)
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    const void *GUIStateEventHandlerCtx,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_SUBOPERATION Suboperation,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
    int32_t  EnrollSampleIndex,
    const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
    const uint8_t  *Text,
    BioAPI_GUI_RESPONSE  *Response,
    int32_t  *EnrollSampleIndexToRecapture);
```

7.71.2.2 Description

This is a function pointer type for an application's GUI event handler function that is to handle GUI state event notification callbacks coming from a BSP through the framework. In order to receive GUI state event notifications, an application shall register a callback function of type **BioAPI_GUI_STATE_EVENT_HANDLER** by providing the callback address of the function, along with a context address, in a call to **BioAPI_SubscribeToGUIEvents** (see 8.3.8).

The framework makes a callback to an application function of this type each time it receives an incoming callback to a function of type **BioSPI_GUI_STATE_EVENT_HANDLER** which it exposes to a BSP. The parameters of the callback (except **GUIStateEventHandlerCtx**) shall be set from the parameters of the incoming callback with the same names; the parameter **GUIStateEventHandlerCtx** shall be set from the GUI state context address originally provided by the application in its call to **BioAPI_SubscribeToGUIEvents**; and the callback address shall be set from the GUI state callback address originally provided by the application in the call to **BioAPI_SubscribeToGUIEvents**.

A BSP can generate GUI state events only during the execution of a call to **BioAPI_Capture**, **BioAPI_Verify**, **BioAPI_Identify**, or **BioAPI_Enroll**. A GUI state event with the moment value **BioAPI_GUI_MOMENT_BEFORE_START** is generated before the start of each suboperation, and a GUI state event with the moment value **BioAPI_GUI_MOMENT_AFTER_END** is generated after the end of the each suboperation (see 7.66).

The main purpose of the GUI state event notification generated before the start of a suboperation is to inform the application that a certain suboperation is ready to start. The main purpose of the GUI state event notification generated after the end of a suboperation is to inform the application about the outcome of the suboperation. The application shall respond to each GUI state event notification by indicating whether the suboperation should start, the suboperation cycle should proceed with the next suboperation, a new suboperation cycle should be started, or the entire operation should be canceled (see 7.69).

The GUI state event handler function and any functions invoked (directly or indirectly) by that function shall not call any BioAPI function.

If the application returns a value different from **`BioAPI_OK`**, the original function will immediately return to the caller with an error code.

7.71.2.3 Parameters

`BSPUuid` (input) – A UUID identifying the BSP to which the GUI event is related.

`UnitID` (input) – The unit ID of the sensor unit of the BSP to which the GUI event is related.

`BSPHandle` (input) – The handle of the BSP attach session to which the GUI event is related.

`GUIStateEventHandlerCtx` (input) – The context address originally provided by the subscriber application as part of the GUI event subscription.

`Operation` (input) – A value indicating the type of operation for which the GUI event has been generated. Only the following values are permitted:

— **`BioAPI_GUI_OPERATION_CAPTURE`**

— **`BioAPI_GUI_OPERATION_VERIFY`**

— **`BioAPI_GUI_OPERATION_IDENTIFY`**

— **`BioAPI_GUI_OPERATION_ENROLL`**

`Suboperation` (input) – A value indicating the type of suboperation for which the GUI event has been generated (the suboperation that is about to start or has just ended). The suboperation shall be one of those that are compatible with Operation (see Table 5).

`Purpose` (input) – This parameter is significant only if **`Suboperation`** is **`BioAPI_GUI_SUBOPERATION_CAPTURE`**, and shall be set to zero otherwise. When significant, it shall indicate the purpose of the capture suboperation (any value of type **`BioAPI_BIR_PURPOSE`**).

`Moment` (input) – A value indicating whether the GUI event has been generated before the start of the suboperation or after the end of the suboperation. Only the following values are permitted:

— **`BioAPI_GUI_MOMENT_BEFORE_START`**

— **`BioAPI_GUI_MOMENT_AFTER_END`**

`ResultCode` (input) – This parameter is significant only if **`Moment`** is **`BioAPI_GUI_MOMENT_AFTER_END`**, and shall be set to zero otherwise. When significant, it shall contain the result code of the suboperation that has just ended. The result code can be either **`BioAPI_OK`** or a BioAPI error code.

`EnrollSampleIndex` (input) – This parameter is significant only if **`Suboperation`** is **`BioAPI_GUI_SUBOPERATION_CAPTURE`**, **`Moment`** is **`BioAPI_GUI_MOMENT_BEFORE_START`**, and a sample for enrollment is about to be captured. Otherwise, it shall be set to zero. When significant, it shall contain the index (an integer from 1 onwards) of the sample to be captured for enrollment by the capture suboperation that is about to start.

`Bitmaps` (input) – An array of bitmaps containing image(s) of the samples captured in the suboperation, which are intended for display to the subject or to an operator. This array usually contains zero or one bitmaps, but may contain multiple bitmaps if the BSP has the ability to capture multiple instances of a biometric modality in a single capture operation.

Text (input) – A textual message, consisting of a string of ISO/IEC 10646 characters encoded in UTF-8, which is intended for display to the subject or to an operator. The content of the message, as well as the language in which it is expressed, are not standardized.

Response (output) – A value indicating the response from the application to the GUI state event notification (see 7.69).

EnrollSampleIndexToRecapture (output) – This parameter is significant only if **Suboperation** is **BioAPI_GUI_SUBOPERATION_CREATETEMPLATE**, **Moment** is **BioAPI_GUI_MOMENT_AFTER_END**, and **Response** is **BioAPI_GUI_RESPONSE_RECAPTURE**. When significant, it shall contain the index (a positive integer) of a sample previously captured for enrollment that the application requests to be recaptured.

EXAMPLE If a BSP provides the enroll type Multiple-Capture to the **BioAPI_GUI_SELECT_EVENT_HANDLER** function, it may capture three biometric samples in each enroll operation. The BSP generates GUI state events before the start and after the end of each capture suboperation (with **EnrollSampleIndex** set to 1, 2, and 3), followed by a GUI state event before the start and after the end of the create template suboperation. When the application receives the GUI state event notification before the start of the create template suboperation, it asks the user which sample is to be replaced. The user chooses the second sample, after reviewing the bitmaps on screen. The application sets **Response** to **BioAPI_GUI_RESPONSE_RECAPTURE** and **EnrollSampleIndexToRecapture** to 2, and returns from the notification callback. The BSP then generates another GUI state event before the start and after the end of a new capture suboperation, with **EnrollSampleIndex** set to 2, and finally another GUI state event before the start and after the end of a new create template suboperation.

NOTE: See also C.7.

7.71.2.4 Return Value

A **BioAPI_RETURN** value indicating success or specifying a particular error condition. The value **BioAPI_OK** indicates success. All other values represent an error condition.

7.71.2.5 Errors

BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED

7.71.3 BioAPI_GUI_PROGRESS_EVENT_HANDLER (BioAPI 2.1)

7.71.3.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioAPI_GUI_PROGRESS_EVENT_HANDLER)
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    const void *GUIProgressEventHandlerCtx,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_SUBOPERATION Suboperation,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_GUI_MOMENT Moment,
    uint8_t SuboperationProgress,
    const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response);
```

7.71.3.2 Description

This is a function pointer type for an application's GUI event handler function that is to handle GUI progress event notification callbacks coming from a BSP through the framework. In order to receive GUI progress event notifications, an application shall register a callback function of type **`BioAPI_GUI_PROGRESS_EVENT_HANDLER`** by providing the callback address of the function, along with a context address, in a call to **`BioAPI_SubscribeToGUIEvents`** (see 8.3.8).

The framework makes a callback to an application function of this type each time it receives an incoming callback to a function of type **`BioSPI_GUI_PROGRESS_EVENT_HANDLER`** which it exposes to a BSP. The parameters of the callback (except **`GUIProgressEventHandlerCtx`**) shall be set from the parameters of the incoming callback with the same names; the parameter **`GUIProgressEventHandlerCtx`** shall be set from the GUI progress context address originally provided by the application in its call to **`BioAPI_SubscribeToGUIEvents`**; and the callback address shall be set from the GUI progress callback address originally provided by the application in the call to **`BioAPI_SubscribeToGUIEvents`**.

A BSP can generate GUI progress events only during the execution of a call to **`BioAPI_Capture`**, **`BioAPI_Process`**, **`BioAPI_CreateTemplate`**, **`BioAPI_VerifyMatch`**, **`BioAPI_IdentifyMatch`**, **`BioAPI_Verify`**, **`BioAPI_Identify`**, or **`BioAPI_Enroll`**. They can be generated at any time, even multiple times, during any suboperation.

The main purpose of the GUI progress event notification is to inform the application about the progress of an lengthy suboperation (such as a capture or an identify match suboperation) as percent of completion, and to send to the application streaming data (as a series of bitmaps) collected by the sensor. One possible use of this information is for the application to show the bitmaps and a progress bar to the subject or an operator. The application shall respond to each GUI progress notification by indicating whether the suboperation should continue or abort (see 7.69).

The GUI progress event handler function and any functions invoked (directly or indirectly) by that function shall not call any BioAPI function.

If the application returns a value different from **`BioAPI_OK`**, the BSP shall cancel the current suboperation and produce an error code. The next GUI event generated by the BSP shall be a GUI state event indicating that the suboperation has ended with that error code.

7.71.3.3 Parameters

`BSPUuid` (input) – A UUID identifying the BSP to which the GUI event is related.

`UnitID` (input) – The unit ID of the sensor unit of the BSP to which the GUI event is related.

`BSPHandle` (input) – The handle of the BSP attach session to which the GUI event is related.

`GUIProgressEventHandlerCtx` (input) – The context address originally provided by the subscriber application as part of the GUI event subscription.

`Operation` (input) – A value indicating the type of operation for which the GUI event has been generated.

`Suboperation` (input) – A value indicating the type of suboperation for which the GUI event has been generated. The suboperation shall be one of those that are compatible with Operation (see Table 5).

`Purpose` (input) – This parameter is significant only if **`Suboperation`** is **`BioAPI_GUI_SUBOPERATION_CAPTURE`**, and shall be set to zero otherwise. When significant, it indicates the purpose of the suboperation (any value of type **`BioAPI_BIR_PURPOSE`**).

`Moment` (input) – A value indicating whether the GUI event has been generated before the start of the suboperation, during the suboperation, or after the end of the suboperation.

`SuboperationProgress` (input) – The percent of completion of the suboperation.

`Bitmaps` (input) – An array of bitmaps containing image(s) of the samples captured in the suboperation, which are intended for display to the subject or to an operator. his array usually contains zero or one bitmaps, but may contain multiple bitmaps if the BSP has the ability to capture multiple instances of a biometric modality in a single capture operation.

`Text` (input) – A textual message, consisting of a string of ISO/IEC 10646 characters encoded in UTF-8, which is intended for display to the subject or to an operator. The content of the message, as well as the language in which it is expressed, are not standardized.

`Response` (output) – A value indicating the response from the application to the GUI select event notification (see 7.69).

NOTE: See also C.7.

7.71.3.4 Return Value

A **`BioAPI_RETURN`** value indicating success or specifying a particular error condition. The value **`BioAPI_OK`** indicates success. All other values represent an error condition.

7.71.3.5 Errors

BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED

2-23) In subclause 7.46, add the following new line after the line containing "BioAPI_SETGUICALLBACKS":

```
#define    BioAPI_SUBSCRIBETOGUIEVENTS     (0x00000002)
```

and add the following text before the note:

The definition BioAPI_SETGUICALLBACKS applies only when the BioAPI version number in use is 2.0. The definition BioAPI_SUBSCRIBETOGUIEVENTS applies only when the BioAPI version number in use is 2.1.

2-24) In subclause 7.47, replace:

If set, the BSP provides GUI streaming data.

with:

If set, the BSP provides GUI streaming data. This definition applies only when the BioAPI version number in use is 2.0.

```
#define BioAPI_GUI_PROGRESS_EVENTS      (0x00000020)
```

If set, the BSP provides GUI progress events. This definition applies only when the BioAPI version number in use is 2.1.

and add the following text at the end of 7.47 (after all previous additions):

```
#define    BioAPI_IDENTIFYINDICATOR    (0x00200000)
```

If set, the BSP supports progress indication during identification. This definition applies only when the BioAPI version number in use is 2.1.

2-25) Replace the heading of subclause 8.3.2 (BioAPI_SetGUICallbacks) with the following text:

8.3.2 BioAPI_SetGUICallbacks (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-26) Add the following text before subclause 8.4:

8.3.3 BioAPI_NotifyGUIProgressEvent (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

```
BioAPI_RETURN BioAPI BioAPI_NotifyGUIProgressEvent
    (const uint8_t *SubscriberEndpointIRI,
    const BioAPI_UUID *GUIEventSubscriptionUuid,
    const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_SUBOPERATION Suboperation,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_GUI_MOMENT Moment,
    uint8_t SuboperationProgress,
    const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response);
```

8.3.3.1 Description

This function enables an application to request that a GUI progress event, generated by the application, be directed to a specified named GUI event subscription. A GUI event passed to this function is called an application-generated GUI event.

When this function is called, the framework shall search all existing named GUI event subscriptions that were created by the application identified by **`SubscriberEndpointIRI`** (**`NULL`** meaning the current application), looking for a subscription where the callback address for the GUI progress event is non-**`NULL`** and the GUI event subscription UUID is the same as **`GUIEventSubscriptionUuid`**. If there is a matching subscription, then the framework shall make a callback to the GUI progress event handler specified in that subscription, setting the input parameters of the callback from the input parameters of the **`BioAPI_NotifyGUIProgressEvent`** call with the same names. After returning from the callback, the framework shall copy the output parameters of the callback to the output parameters of the **`BioAPI_NotifyGUIProgressEvent`** call with the same names.

If no matching named GUI event subscription exists, then the function shall return **`BioAPI_NO_GUI_EVENT_HANDLER`**.

This function shall only be called (for a given BSP UUID) if there is at least one call to **BioAPI_BSPLoad** (for that BSP UUID) for which a corresponding call to **BioAPI_BSPUnload** has not yet been made.

This function is handled internally within the BioAPI Framework and is not passed through to any BSP.

8.3.3.2 Parameters

SubscriberEndpointIRI – An IRI that identifies the application that created one or more named GUI event subscriptions. This shall be **NULL** if that application is the current application. An application can learn the subscriber endpoint IRIs of other applications that are using the framework (and have created one or more named GUI event subscriptions for a given BSP) by calling **BioAPI_QueryGUIEventSubscriptions**. If there are no other applications using the framework, the subscriber endpoint IRI will be **NULL** in all named subscriptions returned by **BioAPI_QueryGUIEventSubscriptions**.

GUIEventSubscriptionUuid – A UUID that identifies a named GUI event subscription. This shall not be **NULL**. In some cases, the application has gotten this UUID from the information returned by a prior call to **BioAPI_QueryGUIEventSubscriptions**; in other cases, the application provides a known UUID.

The other parameters define the GUI progress event generated by the application.

8.3.3.3 Return Value

A **BioAPI_RETURN** value indicating success or specifying a particular error condition. The value **BioAPI_OK** indicates success. All other values represent an error condition.

8.3.3.4 Errors

See **BioAPI Error Handling** (clause 11).

8.3.4 BioAPI_NotifyGUISelectEvent (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

```
BioAPI_RETURN BioAPI BioAPI_NotifyGUISelectEvent
    (const uint8_t *SubscriberEndpointIRI,
    const BioAPI_UUID *GUIEventSubscriptionUuid,
    const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
    int32_t MaxNumEnrollSamples,
    BioAPI_BIR_SUBTYPE_MASK SelectableInstances,
    BioAPI_BIR_SUBTYPE_MASK *SelectedInstances,
    BioAPI_BIR_SUBTYPE_MASK CapturedInstances,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response);
```

8.3.4.1 Description

This function enables an application to request that a GUI select event, generated by the application, be directed to a specified named GUI event subscription. A GUI event passed to this function is called an application-generated GUI event.

When this function is called, the framework shall search all existing named GUI event subscriptions that were created by the application identified by **SubscriberEndpointIRI** (**NULL** meaning the current application), looking for a subscription where the callback address for the GUI select event is non-**NULL** and the GUI event subscription UUID is the same as **GUIEventSubscriptionUuid**. If there is a matching subscription, then the framework shall make a callback to the GUI select event handler specified in that subscription, setting the input parameters of the callback from the input parameters of the **BioAPI_NotifyGUISelectEvent** call with the same names. After returning from the callback, the framework shall copy the output parameters of the callback to the output parameters of the **BioAPI_NotifyGUISelectEvent** call with the same names.

If no matching named GUI event subscription exists, then the function shall return **BioAPI_NO_GUI_EVENT_HANDLER**.

This function shall only be called (for a given BSP UUID) if there is at least one call to **BioAPI_BSPLoad** (for that BSP UUID) for which a corresponding call to **BioAPI_BSPUnload** has not yet been made.

This function is handled internally within the BioAPI Framework and is not passed through to any BSP.

8.3.4.2 Parameters

SubscriberEndpointIRI (input, optional) – An IRI that identifies the application that created one or more named GUI event subscriptions. This shall be **NULL** if that application is the current application. An application can learn the subscriber endpoint IRIs of other applications that are using the framework (and have created one or more named GUI event subscriptions for a given BSP) by calling **BioAPI_QueryGUIEventSubscriptions**. If there are no other applications using the framework, the subscriber endpoint IRI will be **NULL** in all named subscriptions returned by **BioAPI_QueryGUIEventSubscriptions**.

GUIEventSubscriptionUuid (input) – A UUID that identifies a named GUI event subscription. This shall not be **NULL**. In some cases, the application has gotten this UUID from the information returned by a prior call to **BioAPI_QueryGUIEventSubscriptions**; in other cases, the application provides a known UUID.

The other parameters define the GUI select event generated by the application.

8.3.4.3 Return Value

A **BioAPI_RETURN** value indicating success or specifying a particular error condition. The value **BioAPI_OK** indicates success. All other values represent an error condition.

8.3.4.4 Errors

See **BioAPI Error Handling** (clause 11).

8.3.5 BioAPI_NotifyGUIStateEvent (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

```
BioAPI_RETURN BioAPI BioAPI_NotifyGUIStateEvent
    (const uint8_t *SubscriberEndpointIRI,
    const BioAPI_UUID *GUIEventSubscriptionUuid,
    const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_SUBOPERATION Suboperation,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
int32_t  EnrollSampleIndex,
const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
const uint8_t  *Text,
BioAPI_GUI_RESPONSE  *Response,
int32_t  *EnrollSampleIndexToRecapture);
```

8.3.5.1 Description

This function enables an application to request that a GUI state event, generated by the application, be directed to a specified named GUI event subscription. A GUI event passed to this function is called an application-generated GUI event.

When this function is called, the framework shall search all existing named GUI event subscriptions that were created by the application identified by **SubscriberEndpointIRI** (**NULL** meaning the current application), looking for a subscription where the callback address for the GUI state event is non-**NULL** and the GUI event subscription UUID is the same as **GUIEventSubscriptionUuid**. If there is a matching subscription, then the framework shall make a callback to the GUI state event handler specified in that subscription, setting the input parameters of the callback from the input parameters of the **BioAPI_NotifyGUIStateEvent** call with the same names. After returning from the callback, the framework shall copy the output parameters of the callback to the output parameters of the **BioAPI_NotifyGUIStateEvent** call with the same names.

If no matching named GUI event subscription exists, then the function shall return **BioAPI_NO_GUI_EVENT_HANDLER**.

This function shall only be called (for a given BSP UUID) if there is at least one call to **BioAPI_BSPLoad** (for that BSP UUID) for which a corresponding call to **BioAPI_BSPUnload** has not yet been made.

This function is handled internally within the BioAPI Framework and is not passed through to any BSP.

8.3.5.2 Parameters

SubscriberEndpointIRI (input, optional) – An IRI that identifies the application that created one or more named GUI event subscriptions. This shall be **NULL** if that application is the current application. An application can learn the subscriber endpoint IRIs of other applications that are using the framework (and have created one or more named GUI event subscriptions for a given BSP) by calling **BioAPI_QueryGUIEventSubscriptions**. If there are no other applications using the framework, the subscriber endpoint IRI will be **NULL** in all named subscriptions returned by **BioAPI_QueryGUIEventSubscriptions**.

GUIEventSubscriptionUuid (input) – A UUID that identifies a named GUI event subscription. This shall not be **NULL**. In some cases, the application has obtained this UUID from the information returned by a prior call to **BioAPI_QueryGUIEventSubscriptions**; in other cases, the application provides a known UUID.

The other parameters define the GUI state event generated by the application.

8.3.5.3 Return Value

A **BioAPI_RETURN** value indicating success or specifying a particular error condition. The value **BioAPI_OK** indicates success. All other values represent an error condition.

8.3.5.4 Errors

See **BioAPI Error Handling** (clause 11).

8.3.6 BioAPI_QueryGUIEventSubscriptions (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

```
BioAPI_RETURN BioAPI BioAPI_QueryGUIEventSubscriptions
    (const BioAPI_UUID *BSPUuid,
    BioAPI_GUI_EVENT_SUBSCRIPTION **GUIEventSubscriptionList,
    uint32_t *NumberOfElements);
```

8.3.6.1 Description

This function provides information about all existing named GUI event subscriptions for a given BSP. A named GUI event subscription is one that was created by a call to **BioAPI_SubscribeToGUIEvents** specifying a non-**NULL** GUI event subscription UUID. The framework calls the event handlers specified in a named subscription to notify BSP-generated GUI events that have been redirected to that named subscription (see 8.3.7), and to notify application-generated GUI events (see 8.3.3, 8.3.4, and 8.3.5) directed to that named subscription.

By calling this function, an application can learn the subscriber endpoint IRIs of other applications that are using the framework (where this is supported by the implementation architecture of the framework) and have created named GUI event subscriptions for a given BSP. If there are no other applications using the framework, the subscriber endpoint IRIs will be **NULL** in all the named subscriptions returned by this function.

An application can use the information returned by a call to this function (subscriber endpoint IRI, GUI event subscription UUID, and flags) in subsequent calls to **BioAPI_RedirectGUIEvent** (to request that certain subsequent BSP-generated GUI events be redirected to a specified named subscription), or in subsequent calls to **BioAPI_NotifyGUISelectEvent**, **BioAPI_NotifyGUIStateEvent**, etc. (to request that an application-generated GUI event be directed to a specified named subscription).

This function performs the following actions (in order):

a) allocates a memory block large enough to contain an array of elements of type **BioAPI_GUI_EVENT_SUBSCRIPTION** with as many elements as the number of existing named GUI event subscriptions for the given BSP;

b) fills the array with information about the named subscriptions; and

c) returns the address of the array in the **GUIEventSubscriptionList** parameter and the number of elements of the array in the **NumberOfElements** parameter.

This function shall only be called (for a given BSP UUID) if there is at least one call to **BioAPI_BSPLoad** (for that BSP UUID) for which a corresponding call to **BioAPI_BSPUnload** has not yet been made.

This function is handled internally within the BioAPI Framework and is not passed through to any BSP.

The memory block containing the array shall be freed by the application via a call to **BioAPI_Free** (see 8.7.2) when it is no longer needed by the application.

8.3.6.2 Parameters

GUIEventSubscriptionList (output) – A pointer to the address of the array of elements of the type **BioAPI_GUI_EVENT_SUBSCRIPTION** (allocated by the framework) containing the named GUI event subscription information.

NumberOfElements (output) – A pointer to the number of elements of the array.

8.3.6.3 Return Value

A **BioAPI_RETURN** value indicating success or specifying a particular error condition. The value **BioAPI_OK** indicates success. All other values represent an error condition.

8.3.6.4 Errors

See **BioAPI Error Handling** (clause 11).

8.3.7 BioAPI_RedirectGUIEvents (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

```
BioAPI_RETURN BioAPI BioAPI_RedirectGUIEvents
    (const uint8_t *SubscriberEndpointIRI,
    const BioAPI_UUID *GUIEventSubscriptionUuid,
    BioAPI_HANDLE BSPHandle,
    BioAPI_BOOL GUISelectEventRedirected,
    BioAPI_BOOL GUIStateEventRedirected,
    BioAPI_BOOL GUIProgressEventRedirected);
```

8.3.7.1 Description

This function creates a "GUI event redirector" for the current application. GUI event redirectors are maintained by the framework at runtime (usually in an internal table), and enable an application to delegate the handling of BSP-generated GUI events either to another application that is using the framework (where this is supported by the implementation architecture of the framework), or to another component of the same application.

An application calling this function shall provide the subscriber endpoint IRI of the application that created the named GUI event subscription (**NULL** if it is the same application) and the GUI event subscription UUID of the subscription, as well as information specifying the scope of the GUI event redirector (what BSP attach session and which type(s) of GUI events are affected). All the parameter values provided in a call to this function shall become part of the GUI event redirector maintained by the framework.

Each call to this function establishes a new GUI event redirector, and does not modify or replace an existing redirector. GUI event redirectors specifying a given BSP attach session handle shall be destroyed automatically by the framework when the attach session is destroyed. At any time, the application can ask the framework to destroy an existing redirector by making a call to **BioAPI_UnredirectGUIEvents** providing the same parameters it provided in the corresponding call to **BioAPI_RedirectGUIEvents**.

Multiple call to this function shall not result, at any time, in multiple GUI select event redirectors, multiple GUI state event redirectors, or multiple GUI progress event redirectors established for the same BSP handle. This is to ensure that, for any incoming GUI event, there will be no ambiguity as to how to redirect the event.

NOTE: This implies that two redirectors with the same parameters are not allowed.

When the framework receives a call to this function that successfully creates a GUI event redirector, it shall call in turn the function **BioSPI_SubscribeToGUIEvents** of the BSP if and only if there were no GUI event subscriptions and no GUI event redirectors already established for that BSP before this call. Subsequent calls to **BioAPI_RedirectGUIEvents** will not result in further calls to **BioSPI_SubscribeToGUIEvents** unless all existing redirectors and subscriptions for that BSP have been destroyed.

When the framework receives a GUI select event notification callback from a BSP, it shall first search all existing GUI event redirectors, looking for a redirector where GUISelectEventRedirected has the value TRUE and the BSP handle is the same as the one in the GUI event. One of the two following subclauses applies.

If there is no matching redirector, then the framework shall search all anonymous GUI event subscriptions created by the application that created the BSP attach session, looking for a subscription with a non-**NULL** callback address for the GUI select event and with either a BSP handle matching the BSP handle of the event or a BSP UUID matching the BSP UUID of the event. If there is a matching subscription, the framework shall call the GUI select event handler in that subscription, otherwise it shall return control to the original callback from the BSP with default output parameters.

If there is a matching redirector, then the framework shall search all existing named GUI event subscriptions that were created by the application identified by the subscriber endpoint IRI in the redirector (**NULL** meaning the current application), looking for a subscription with a non-**NULL** callback address for the GUI select event, with a GUI event subscription UUID matching the GUI event subscription UUID in the redirector, and with a BSP UUID matching the BSP UUID of the event. If there is a matching subscription, then the framework shall make a callback to the GUI select event handler specified in that subscription, otherwise it shall return control to the original callback from the BSP with default output parameters.

In both cases, after returning from the application callback, the framework shall copy the output parameters of the application callback to the output parameters of the framework callback with the same names.

The four subclauses above apply in the same way to GUI state and to GUI progress events.

This function shall only be called (for a given BSP UUID) if there is at least one call to **BioAPI_BSPLoad** (for that BSP UUID) for which a corresponding call to **BioAPI_BSPUnload** has not yet been made.

This function is handled internally within the BioAPI Framework and is not passed through to any BSP.

8.3.7.2 Parameters

SubscriberEndpointIRI (input, optional) – An IRI that identifies the application that created a named GUI event subscription. This shall be **NULL** if the subscriber application is the same as the current application.

GUIEventSubscriptionUuid (input) – A UUID that identifies a named GUI event subscription. In some cases, an application has gotten this UUID from the information returned by a prior call to **BioAPI_QueryGUIEventSubscriptions**. In other cases, the application provides a known UUID.

BSPHandle (input) – The handle of a BSP attach session to which the redirector is limited. GUI events related to other BSP attach sessions will not be affected by this call.

GUISelectEventRedirected (input) – Flag indicating whether GUI select events are in the scope of the redirector. GUI select events will be affected by this call if and only if this flag has the value TRUE.

GUIStateEventRedirected (input) – Flag indicating whether GUI state events are in the scope of the redirector. GUI state events will be affected by this call if and only if this flag has the value TRUE.

GUIProgressEventRedirected (input) – Flag indicating whether GUI progress events are in the scope of the redirector. GUI progress events will be affected by this call if and only if this flag has the value TRUE.

8.3.7.3 Return Value

A **BioAPI_RETURN** value indicating success or specifying a particular error condition. The value **BioAPI_OK** indicates success. All other values represent an error condition.

8.3.7.4 Errors

See **BioAPI Error Handling** (clause 11).

8.3.8 BioAPI_SubscribeToGUIEvents (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

```
BioAPI_RETURN BioAPI BioAPI_SubscribeToGUIEvents
    (const BioAPI_UUID *GUIEventSubscriptionUuid,
    const BioAPI_UUID *BSPUuid,
    const BioAPI_HANDLE *BSPHandle,
    BioAPI_GUI_SELECT_EVENT_HANDLER GUISelectEventHandler,
    const void *GUISelectEventHandlerCtx,
    BioAPI_GUI_STATE_EVENT_HANDLER GUIStateEventHandler,
    const void *GUIStateEventHandlerCtx,
    BioAPI_GUI_PROGRESS_EVENT_HANDLER GUIProgressEventHandler,
    const void *GUIProgressEventHandlerCtx);
```

8.3.8.1 Description

This function creates a "GUI event subscription" for the current application. GUI event subscriptions are maintained by the framework at runtime (usually as entries of an internal table), and enable an application to receive GUI event notifications at certain specified callback addresses.

The application shall provide the callback addresses of one to three GUI event handlers (a GUI select event handler, a GUI state event handler and a GUI progress event handler), as well as information specifying the scope of the subscription (either a loaded BSP or an attach session of a BSP), thus limiting the GUI event notifications that the framework will send to those GUI event handlers. All the parameter values provided in a call to this function shall become part of the GUI event subscription maintained by the framework.

In any call to this function, either a BSP UUID or a BSP handle (but not both) shall be provided. If a BSP handle is provided, the subscription is limited to GUI event notifications that carry that BSP handle. If a BSP UUID is provided, the subscription is limited to GUI event notifications that carry that BSP UUID, and may or may not also carry a BSP handle. (All GUI event notification that carry a BSP handle also carry the UUID of the BSP, but not vice versa.)

Each call to this function establishes a new GUI event subscription, and does not modify or replace an existing subscription. GUI event subscriptions specifying a BSP attach session handle shall be destroyed automatically by the framework when the attach session is destroyed. GUI event subscriptions specifying a BSP UUID shall be destroyed automatically by the framework when the BSP is unloaded. At any time, the application can ask the framework to destroy an existing subscription by making a call to **BioAPI_UnsubscribeFromGUIEvents** providing the same parameters that were provided in the corresponding call to **BioAPI_SubscribeToGUIEvents**.

The context addresses provided in the call shall be passed back by the framework to the application in subsequent callbacks to the event handlers. This International Standard does not assign any meaning to the context addresses, but specifies them to satisfy the needs of some applications.

Multiple call to this function shall not result, at any time, in multiple GUI select event handlers, multiple GUI state event handlers, or multiple GUI progress event handlers, established for the same BSP handle or for the same BSP UUID. This includes the case in which one subscription specifies a BSP UUID and another subscription specifies the handle of an attach session of the same BSP. This is to ensure that, for any incoming GUI event, there will be no ambiguity as to which GUI event handler (if any) is to be called by the framework.

NOTE: This implies that two subscriptions with the same parameters are not allowed.

When the framework receives a call to this function that successfully creates a GUI event subscription for a given BSP, it shall call in turn the function **BioSPI_SubscribeToGUIEvents** of the BSP if and only if there were no GUI event subscriptions and no GUI event redirectors (see 8.3.7) already established for that BSP before this call. Subsequent calls to **BioAPI_SubscribeToGUIEvents** for the same BSP will not result in further calls to **BioSPI_SubscribeToGUIEvents** unless all existing subscriptions and redirectors for that

BSP have been destroyed. A BSP does not need to know how many GUI event subscriptions the application is requesting for it, nor whether those subscriptions specify a BSP UUID or a BSP handle, nor the types, callback addresses, and context addresses of the GUI event handlers that are specified in those subscriptions.

This function shall only be called (for a given BSP UUID) if there is at least one call to **`BioAPI_BSPLoad`** (for that BSP UUID) for which a corresponding call to **`BioAPI_BSPUnload`** has not yet been made.

This function is handled internally within the BioAPI Framework and is not passed through to any BSP.

NOTE: See also C.7.

8.3.8.2 Parameters

`GUIEventSubscriptionUuid` (input, optional) – The identifier of the subscription. This parameter shall be **`NULL`** (the usual case) for an anonymous GUI event subscription (one concerning the reception of GUI events generated by a BSP and not redirected). This parameter shall be non-**`NULL`** for a named subscription (one concerning the reception of redirected GUI event notifications and application-generated GUI event notifications).

`BSPUuid` (input, optional) – A UUID identifying the BSP to which the subscription is limited.

`BSPHandle` (input, optional) – The handle of a BSP attach session to which the subscription is limited. This shall be **`NULL`** if **`GUIEventSubscriptionUuid`** is not **`NULL`** (a named subscription cannot specify an attach session handle).

`GUISelectEventHandler` (input, optional) – The callback address of an application function that is to receive GUI select event notifications from the framework.

`GUISelectEventHandlerCtx` (input) – A context address that is to be passed back by the framework to the application on each callback to the GUI select event handler.

`GUIStateEventHandler` (input, optional) – The callback address of an application function that is to receive GUI state event notifications from the framework.

`GUIStateEventHandlerCtx` (input) – A context address that is to be passed back by the framework to the application on each callback to the GUI state event handler.

`GUIProgressEventHandler` (input, optional) – The callback address of an application function that is to receive GUI progress event notifications from the framework.

`GUIProgressEventHandlerCtx` (input) – A context address that is to be passed back by the framework to the application on each callback to the GUI progress event handler.

Exactly one of **`BSPUuid`** and **`BSPHandle`** shall be specified. At least one of **`GUISelectEventHandler`**, **`GUIStateEventHandler`**, and **`GUIProgressEventHandler`** shall be specified. A context address shall only be specified if the corresponding callback address is specified. The parameters that are not specified shall be set to **`NULL`**.

8.3.8.3 Return Value

A **`BioAPI_RETURN`** value indicating success or specifying a particular error condition. The value **`BioAPI_OK`** indicates success. All other values represent an error condition.

8.3.8.4 Errors

BioAPIERR_INVALID_POINTER

See also **BioAPI Error Handling** (clause 11).

8.3.9 BioAPI_UnredirectGUIEvents (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

```
BioAPI_RETURN BioAPI BioAPI_UnredirectGUIEvents
    (const uint8_t *SubscriberEndpointIRI,
    const BioAPI_UUID *GUIEventSubscriptionUuid,
    BioAPI_HANDLE BSPHandle,
    BioAPI_BOOL GUISelectEventRedirected,
    BioAPI_BOOL GUIStateEventRedirected,
    BioAPI_BOOL GUIProgressEventRedirected);
```

8.3.9.1 Description

This function destroys a GUI event redirector previously created by a call to **BioAPI_RedirectGUIEvents**. The particular redirector that is destroyed is the one whose definition exactly matches the parameters of the call.

NOTE: A call to this function will not destroy a redirector, even if there is only one redirector, unless the parameters of the call have exactly the same values as those of the call to **BioAPI_RedirectGUIEvents** that created the redirector. The application needs to remember the value of those parameters.

When the framework receives a call to this function that successfully destroys a GUI event redirector, it shall call in turn the function **BioSPI_UnsubscribeFromGUIEvents** of the BSP if and only if there are no more GUI event subscriptions and no more GUI event redirectors established for that BSP.

Usually, applications will not need to call this function, because all GUI event redirectors for a given BSP attach session are automatically destroyed by the framework when the attach session is destroyed. This function can be used when the application needs to modify the current GUI event redirections but does not want to destroy an attach session.

This function shall only be called (for a given BSP UUID) if there is at least one call to **BioAPI_BSPLoad** (for that BSP UUID) for which a corresponding call to **BioAPI_BSPUnload** has not yet been made.

This function is handled internally within the BioAPI Framework and is not passed through to any BSP.

NOTE: See also C.7.

8.3.9.2 Parameters

The parameters of this function have the same meaning as the parameters of the function **BioAPI_RedirectGUIEvents** with the same names, and are used by the framework to identify the existing GUI event redirector that is to be destroyed.

8.3.9.3 Return Value

A BioAPI_RETURN value indicating success or specifying a particular error condition. The value BioAPI_OK indicates success. All other values represent an error condition.

8.3.9.4 Errors

BioAPIERR_INVALID_POINTER

See also **BioAPI Error Handling** (clause 11).

8.3.10 BioAPI_UnsubscribeFromGUIEvents (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

```
BioAPI_RETURN BioAPI BioAPI_UnsubscribeFromGUIEvents
    (const BioAPI_UUID *GUIEventSubscriptionUuid,
    const BioAPI_UUID *BSPUuid,
    const BioAPI_HANDLE *BSPHandle,
    BioAPI_GUI_SELECT_EVENT_HANDLER GUISelectEventHandler,
    const void *GUISelectEventHandlerCtx,
    BioAPI_GUI_STATE_EVENT_HANDLER GUIStateEventHandler,
    const void *GUIStateEventHandlerCtx,
    BioAPI_GUI_PROGRESS_EVENT_HANDLER GUIProgressEventHandler
    const void *GUIProgressEventHandlerCtx);
```

8.3.10.1 Description

This function destroys a GUI event subscription previously created by a call to **BioAPI_SubscribeToGUIEvents**. The particular subscription that is destroyed is the one whose definition exactly matches the parameters of the call.

NOTE: A call to this function will not destroy a GUI event subscription, even if there is only one subscription, unless the parameters of the call have exactly the same values as those of the call to **BioAPI_SubscribeToGUIEvents** that created the subscription. The application needs to remember the value of those parameters.

When the framework receives a call to this function that successfully destroys a GUI event subscription, it shall call in turn the function **BioSPI_UnsubscribeFromGUIEvents** of the BSP if and only if there are no more GUI event subscriptions and no more GUI event redirectors (see 8.3.7) established for that BSP.

Usually, applications will not need to call this function, because all GUI event subscriptions for a given BSP UUID or BSP attach session handle are automatically destroyed by the framework when the attach session is destroyed or when the BSP is unloaded. This function can be used when the application needs to modify the current GUI event subscriptions but does not want to unload a BSP or destroy an attach session.

EXAMPLE This function can be used in the following cases: (1) an application uses the application-controlled GUI both at enrollment and at verification/identification, but with different GUI event handlers; (2) an application uses the BSP-controlled GUI at enrollment but the application-controlled GUI at verification/identification, or vice versa.

This function shall only be called (for a given BSP UUID) if there is at least one call to **BioAPI_BSPLoad** (for that BSP UUID) for which a corresponding call to **BioAPI_BSPUnload** has not yet been made.

This function is handled internally within the BioAPI Framework and is not passed through to any BSP.

NOTE: See also C.7.

8.3.10.2 Parameters

The parameters of this function have the same meaning as the parameters of the function **BioAPI_SubscribeToGUIEvents** with the same names, and are used by the framework to identify the existing GUI event subscription that is to be destroyed.

8.3.10.3 Return Value

A **BioAPI_RETURN** value indicating success or specifying a particular error condition. The value **BioAPI_OK** indicates success. All other values represent an error condition.

8.3.10.4 Errors

BioAPIERR_INVALID_POINTER

See also **BioAPI Error Handling** (clause 11).

2-27) Replace the parameter definitions in 9.2.1.1 with the following text:

BSPUuid (input) – The UUID of the biometric service provider raising the event.

UnitID (input) – The ID of the BioAPI Unit associated with the event.

UnitSchema (input) – A pointer to the unit schema of the BioAPI Unit associated with the event.

EventType (input) – The **BioAPI_EVENT** that has occurred.

2-28) Add the following text before subclause 9.3:

9.2.4 BioSPI_GUI_PROGRESS_EVENT_HANDLER (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

9.2.4.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioSPI_GUI_PROGRESS_EVENT_HANDLER)
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_SUBOPERATION Suboperation,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_GUI_MOMENT Moment,
    uint8_t SuboperationProgress,
    const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response);
```

9.2.4.2 Description

This is a function pointer type for a framework's GUI event handler function that is to handle GUI progress event notification callbacks coming from a BSP. In order to receive GUI progress event notifications, the framework shall register a callback function of type **BioSPI_GUI_PROGRESS_EVENT_HANDLER** by providing the callback address of the function in a call to **BioSPI_SubscribeToGUIEvents** (see 9.3.3.3).

The BSP makes a callback to a framework function of this type in order to notify to the framework (and ultimately to the application) a GUI progress event generated while executing a BioSPI function.

A BSP can generate GUI progress events only during the execution of a call to **BioSPI_Capture**, **BioSPI_Process**, **BioSPI_CreateTemplate**, **BioSPI_VerifyMatch**, **BioSPI_IdentifyMatch**, **BioSPI_Verify**, **BioSPI_Identify**, or **BioSPI_Enroll**. They can be generated at any time, even multiple times, during any suboperation.

9.2.4.3 Parameters

The parameters of this function pointer type are the same as those of **BioAPI_GUI_PROGRESS_EVENT_HANDLER**, except for the absence of the context address parameter.

NOTE: See also C.7.

9.2.4.4 Return Value

A **BioAPI_RETURN** value indicating success or specifying a particular error condition. The value **BioAPI_OK** indicates success. All other values represent an error condition.

9.2.4.5 Errors

BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED

9.2.5 BioSPI_GUI_SELECT_EVENT_HANDLER (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

9.2.5.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioSPI_GUI_SELECT_EVENT_HANDLER)
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    BioAPI_GUI_ENROLL_TYPE EnrollType,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
    int32_t MaxNumEnrollSamples,
    BioAPI_BIR_SUBTYPE_MASK SelectableInstances,
    BioAPI_BIR_SUBTYPE_MASK *SelectedInstances,
    BioAPI_BIR_SUBTYPE_MASK CapturedInstances,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response);
```

9.2.5.2 Description

This is a function pointer type for a framework's GUI event handler function that is to handle GUI select event notification callbacks coming from a BSP. In order to receive GUI select event notifications, the framework shall register a callback function of type **BioSPI_GUI_SELECT_EVENT_HANDLER** by providing the callback address of the function in a call to **BioSPI_SubscribeToGUIEvents** (see 9.3.3.3).

The BSP makes a callback to a framework function of this type in order to notify to the framework (and ultimately to the application) each GUI select event generated while executing a BioSPI function.

A BSP can generate GUI select events only during the execution of a call to **BioSPI_Capture**, **BioSPI_Verify**, **BioSPI_Identify**, or **BioSPI_Enroll**. A GUI select event with the moment value **BioAPI_GUI_MOMENT_BEFORE_START** is generated before the start of each suboperation cycle, and a GUI select event with the moment value **BioAPI_GUI_MOMENT_AFTER_END** is generated after the end of each suboperation cycle (see 7.66).

9.2.5.3 Parameters

The parameters of this function pointer type are the same as those of **`BioAPI_GUI_SELECT_EVENT_HANDLER`**, except for the absence of the context address parameter.

NOTE: See also C.7.

9.2.5.4 Return Value

A **`BioAPI_RETURN`** value indicating success or specifying a particular error condition. The value **`BioAPI_OK`** indicates success. All other values represent an error condition.

9.2.5.5 Errors

BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED

9.2.6 BioSPI_GUI_STATE_EVENT_HANDLER (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

9.2.6.1 Callback function

```
typedef BioAPI_RETURN (BioAPI *BioSPI_GUI_STATE_EVENT_HANDLER)
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_SUBOPERATION Suboperation,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
    int32_t  EnrollSampleIndex,
    const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
    const uint8_t  *Text,
    BioAPI_GUI_RESPONSE  *Response,
    int32_t  *EnrollSampleIndexToRecapture);
```

9.2.6.2 Description

This is a function pointer type for a framework's GUI event handler function that is to handle GUI state event notification callbacks coming from a BSP. In order to receive GUI state event notifications, the framework shall register a callback function of type **`BioSPI_GUI_STATE_EVENT_HANDLER`** by providing the callback address of the function in a call to **`BioSPI_SubscribeToGUIEvents`** (see 9.3.3.3).

The BSP makes a callback to a framework function of this type in order to notify to the framework (and ultimately to the application) each GUI state event generated while executing a BioSPI function.

A BSP can generate GUI state events only during the execution of a call to **`BioSPI_Capture`**, **`BioSPI_Verify`**, **`BioSPI_Identify`**, or **`BioSPI_Enroll`**. A GUI state event with the moment value **`BioAPI_GUI_MOMENT_BEFORE_START`** is generated before the start of each suboperation, and a GUI state event with the moment value **`BioAPI_GUI_MOMENT_AFTER_END`** is generated after the end of each suboperation (see 7.66).

9.2.6.3 Parameters

The parameters of this function pointer type are the same as those of **`BioAPI_GUI_STATE_EVENT_HANDLER`**, except for the absence of the context address parameter.

NOTE: See also C.7.

9.2.6.4 Return Value

A **`BioAPI_RETURN`** value indicating success or specifying a particular error condition. The value **`BioAPI_OK`** indicates success. All other values represent an error condition.

9.2.6.5 Errors

BioAPIERR_USER_CANCELLED
BioAPIERR_FUNCTION_FAILED

2-29) Replace the heading of subclause 9.3.3.2 (BioSPI_SetGUICallbacks) with the following text:

9.3.3.2 BioSPI_SetGUICallbacks (BioAPI 2.0)

This subclause applies only when the BioAPI version number in use is 2.0.

2-30) Add the following text after subclause 9.3.3.2:

9.3.3.3 BioSPI_SubscribeToGUIEvents (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

```
BioAPI_RETURN BioAPI BioSPI_SubscribeToGUIEvents
    (const BioAPI_UUID *BSPUuid,
    BioSPI_GUI_SELECT_EVENT_HANDLER FwGUISelectEventHandler,
    BioSPI_GUI_STATE_EVENT_HANDLER FwGUIStateEventHandler,
    BioSPI_GUI_PROGRESS_EVENT_HANDLER FwGUIProgressEventHandler);
```

This function provides to the BSP the callback addresses of the framework's GUI select event handler, GUI state event handler, and GUI progress event handler. The three addresses shall not be **`NULL`**.

After a call to this function and until a subsequent call to **`BioSPI_UnsubscribeFromGUIEvents`** or **`BioSPI_BSPUnload`**, for each GUI event that the BSP generates while executing BioSPI function calls, the BSP shall notify that event to the framework by calling back the framework GUI event handler corresponding to the type of GUI event. In addition, any BSP-controlled GUI shall be disabled during that time.

The parameter BSPUuid is provided only as a reliability measure. The BSP shall ensure that the value of this parameter matches its BSP product UUID.

Unlike the function **`BioAPI_SubscribeToGUIEvents`**, this function shall not create a new subscription on each call. On each call, the BSP shall simply replace the old callback addresses (initially **`NULL`**) with the ones provided in the call.

A single call to **`BioAPI_UnsubscribeFromGUIEvents`** will clear the three callback addresses. Only one such call is needed even if the function **`BioAPI_SubscribeToGUIEvents`** has been called multiple times.

NOTE: Once the framework has called this function, it will never call it again until after a call to **BioSPI_UnsubscribeFromGUIEvents** or **BioSPI_BSPUnload**. The framework will pass the same three callback addresses every time it calls this function.

9.3.3.3.1 Parameters

BSPUuid (input) – A UUID identifying the BSP.

FwGUISelectEventHandler (input) – The callback address of a framework function that is to receive GUI select event notifications from the BSP.

FwGUIStateEventHandler (input) – The callback address of a framework function that is to receive GUI state event notifications from the BSP.

FwGUIProgressEventHandler (input, optional) – The callback address of a framework function that is to receive GUI progress event notifications from the BSP.

9.3.3.4 BioSPI_UnsubscribeFromGUIEvents (BioAPI 2.1)

This subclause applies only when the BioAPI version number in use is 2.1.

```
BioAPI_RETURN BioAPI BioSPI_UnsubscribeFromGUIEvents
    (const BioAPI_UUID *BSPUuid);
```

This function clears the callback addresses of the framework's GUI select event handler, GUI state event handler, and GUI progress event handler in the BSP.

After a call to this function, the BSP shall cease to notify GUI events to the framework.

The parameter **BSPUuid** is provided only as a reliability measure. The BSP shall ensure that the value of this parameter matches its BSP product UUID.

9.3.3.4.1 Parameters

BSPUuid (input) – A UUID identifying the BSP.

NOTE 2: See also C.7.

2-31) Add the following text at the end of subclause 11.2.7:

#define BioAPIERR_TOO_EARLY (0x000502)

The behavior of the subject was too early. This definition applies only when the BioAPI version number in use is 2.1.

#define BioAPIERR_TOO_LATE (0x000503)

The behavior of the subject was too late. This definition applies only when the BioAPI version number in use is 2.1.

#define BioAPIERR_TOO_FAST (0x000504)

The behavior of the subject was too fast. This definition applies only when the BioAPI version number in use is 2.1.

#define BioAPIERR_TOO_SLOW (0x000505)

The behavior of the subject was too slow. This definition applies only when the BioAPI version number in use is 2.1.

#define BioAPIERR_TOO_LONG (0x000506)

The captured data was too long. This definition applies only when the BioAPI version number in use is 2.1.

#define BioAPIERR_TOO_SHORT (0x000507)

The captured data was too short. This definition applies only when the BioAPI version number in use is 2.1.

#define BioAPIERR_TOO_LARGE (0x000508)

The captured data was too large. This definition applies only when the BioAPI version number in use is 2.1.

#define BioAPIERR_TOO_SMALL (0x000509)

The captured data was too small. This definition applies only when the BioAPI version number in use is 2.1.

2-32) Modify Table 3 in 10.1.2 as follows

Replace the first row of the table with the following two rows:

BSPUUID	UINT8_T [16]	UUID uniquely identifying the BSP. This field shall be present only when the BioAPI version number in use is 2.0.
BSPProductUUID	UINT8_T [16]	UUID uniquely identifying the BSP as a software product. This field shall be present only when the BioAPI version number in use is 2.1.

and add the following rows at the end of the table:

MaxNumEnrollInstances	UINT32_T	The maximum number of distinct instances that a BSP can create reference templates in one enroll operation. This field shall be present only when the BioAPI version number in use is 2.1.
HostingEndpointIRI	UINT8_T	An IRI identifying the framework whose component registry contains a registration of the BSP. This field shall be present only when the BioAPI version number in use is 2.1.
BSPAccessUUID	BioAPI_UUID	A UUID, unique within the scope of an application, which the application may use to refer to the BSP as an alternative to the BSP product UUID. This field shall be present only when the BioAPI version number in use is 2.1.

2-33) Add the following text at the end of 11.2.3:

#define BioAPIERR_IDENTIFY_IN_PROGRESS (0x000120)

Identification is in progress. This definition applies only when the BioAPI version number in use is 2.1.

#define BioAPIERR_LOW_QUALITY_REFERENCE_TEMPLATE (0x000121)

Quality of reference template is low. This definition applies only when the BioAPI version number in use is 2.1.

```
#define BioAPIERR_NO_GUI_EVENT_HANDLER  (0x000122)
```

GUI event handler is not set. This definition applies only when the BioAPI version number in use is 2.1.

2-34) Replace A.1.1 with the following text:

A.1.1 Conformance to this part of ISO/IEC 19784 falls into the following five classes:

- BioAPI conformant biometric application
- BioAPI 2.0 conformant BioAPI framework
- BioAPI 2.1 conformant BioAPI framework
- BioAPI 2.0 conformant BSP, comprising one of the following subclasses:
    - BioAPI 2.0 conformant Verification BSP
    - BioAPI 2.0 conformant Identification BSP
    - BioAPI 2.0 conformant Capture BSP
    - BioAPI 2.0 conformant Verification Engine
    - BioAPI 2.0 conformant Identification Engine
- BioAPI 2.1 conformant BSP, comprising one of the following subclasses:
    - BioAPI 2.1 conformant Verification BSP
    - BioAPI 2.1 conformant Identification BSP
    - BioAPI 2.1 conformant Capture BSP
    - BioAPI 2.1 conformant Verification Engine
    - BioAPI 2.1 conformant Identification Engine

2-35) Replace A.3.3 with the following text:

A.3.3 A conformant BioAPI 2.0 Framework is required to support all the mandatory and optional provisions of this International Standard, except the subclauses and definitions that are specified as applying to version 2.1 only.

A.3.4 A conformant BioAPI 2.1 Framework is required to support all the mandatory and optional provisions of this International Standard, except the subclauses and definitions that are specified as applying to version 2.0 only.

A.3.5 No subsets of conformant Framework functions are defined.

2-36) In A.3.2, add the following text after "To claim conformance to the BioAPI specification":

(either version 2.0 or version 2.1)

2-37) In the first paragraph of A.4, add the following text after "To claim conformance to the BioAPI specification":

(either version 2.0 or version 2.1)

2-38) Add the following new paragraphs after the first paragraph of A.4:

BioAPI 2.0 conformant BSPs (of any conformance subclass) are required to support all the mandatory provisions of this International Standard according to their conformance subclass, except the subclauses and definitions that are specified as applying to version 2.1 only.

BioAPI 2.1 conformant BSPs (of any conformance subclass) are required to support all the mandatory provisions of this International Standard according to their conformance subclass, except the subclauses and definitions that are specified as applying to version 2.0 only.

2-39) Modify Table A.1 as follows:

Replace the row containing "`BioSPI_SetGUICallbacks`" with the following rows:

`BioSPI_SetGUICallbacks` (BioAPI 2.0)					
`BioSPI_SubscribeToGUIEvents` (BioAPI 2.1)					
`BioSPI_UnsubscribeFromGUIEvents` (BioAPI 2.1)					

and add the following rows at the end of the table:

GUI events					
GUI select events (BioAPI 2.1)					
GUI state events (BioAPI 2.1)					
GUI progress events (BioAPI 2.1)					

2-40) In Table A.2, replace the row containing "GUI streaming callbacks" with the following rows:

GUI streaming callbacks (BioAPI 2.0)		
Generation of GUI progress events during capture suboperations (BioAPI 2.1)		
Generation of GUI progress events during identify match suboperations (BioAPI 2.1)		

2-41) In the table in B.10, replace the row containing "BioAPI_BIR_BIOMETRIC_TYPE" with the following row:

BioAPI_BIR_BIOMETRIC_TYPE (BioAPI 7.8)	CBEFF_BDB_biometric_type	4	NO BIOTYPE AVAILABLE MULTIPLE_BIOMETRIC_TYPES FACE VOICE FINGER IRIS RETINA HAND GEOMETRY SIGNATURE SIGN KEYSTROKE LIP MOVEMENT RESERVED RESERVED GAIT VEIN DNA EAR FOOT SCENT OTHER PASSWORD	'00 00 00 00' '00 00 00 01' '00 00 00 02' '00 00 00 04' '00 00 00 08' '00 00 00 10' '00 00 00 20' '00 00 00 40' '00 00 00 80' '00 00 01 00' '00 00 02 00' '00 00 04 00' '00 00 08 00' '00 00 10 00' '00 00 20 00' '00 00 40 00' '00 00 80 00' '00 01 00 00' '00 02 00 00' '40 00 00 00' '80 00 00 00'	

2-42) In the table in B.10, replace the row containing "BioAPI_BIR_SUBTYPE" with the following row:

BioAPI_BIR_SUBTYPE (BioAPI 7.14)	CBEFF_BDB_biometric_subtype	1	NO_SUBTYPE_AVAILABLE VEIN_ONLY_MASK LEFT_MASK RIGHT_MASK THUMB POINTERFINGER MIDDLEFINGER RINGFINGER LITTLEFINGER VEIN_PALM VEIN_BACKOFHAND VEIN_WRIST	'00' '80' '01' '02' '04' '08' '10' '20' '40' '04' '08' '10'	

2-43) Replace subclause C.7 with the following text:

C.7 User Interface Considerations (BioAPI 2.1)

The considerations and the examples in this subclause (C.7) apply only when the BioAPI version in use is 2.1.

C.7.1 General

A user interface for passwords and PINs is straightforward, but a user interface for biometrics can be quite complex and technology dependent, requiring multiple interactions with the subject providing a biometric sample. Some current implementations of some biometric technologies present streams of data to the subject (face images and voice, for example), while others require the subject to validate each biometric sample taken (face, voice, or signature, for example). During enrollment, some technologies take multiple biometric samples and verify each sample against the previous samples (test-verification), using some matching algorithm. The number of biometric samples taken for a particular purpose (enrollment, verification, or identification) may vary from technology to technology and from implementation to implementation. Finally, a user interface for enrollment is generally different from a user interface for verification or identification, the former generally involving more interactions and being more time-consuming, and the latter being as simple and fast as possible. Thus standardisation of user interfaces for biometric enrollment, verification or identification poses significant challenges if standardised mechanisms or sufficient flexibility are to be provided. The BioAPI GUI interface provides that standard, with appropriate flexibility.

Most current BSP implementations come with a built-in user interface (indeed this is a BioAPI requirement for a conforming BSP implementation), so the application need not be aware of any detailed interaction between the BSP and the subject providing the biometric sample. However, BioAPI also enables the application to control the “look and feel” of the user interface by allowing an application to provide a GUI event handler that the BioAPI framework will call during certain operations, in response to a callback from a BSP. Some of the advantages of an application-controlled GUI over a BSP-controlled GUI are:

a) the application can use graphics and text (perhaps in a separate window) to provide feedback, challenges, or instructions to the subject regarding the ongoing biometric operation;

b) the graphics can be displayed in a way that is consistent with the look and feel of the specific application and can be included inside the application’s main window, or in a separate window, as an application choice; and

c) the application can use a human language that may not be supported by the BSP as (if an application-controlled GUI is in use) most callback interactions between the BSP and the application (through the BioAPI framework) merely identify the semantics to be presented to the user; the human language used for the presentation of the message is a matter for the application (although the GUI progress event also allows the BSP to provide a grey-scale or colored picture).

GUI select and state event notifications are used to control the gathering of samples and to indicate state changes (within an enroll, verify, or identify operation) to the application. An application calls the function **`BioAPI_SubscribeToGUIEvents`** in order to control the GUI itself instead of letting the BSP control the GUI.

GUI select and state events are generated by a BSP only while the BSP is executing an operation invoked by an application's call to any of the following BioAPI functions:

— **`BioAPI_Capture`**

— **`BioAPI_Enroll`**

— **`BioAPI_Verify`**

— **`BioAPI_Identify`**

GUI progress events can be generated by a BSP while the BSP is executing an operation invoked by an application's call to any of the following BioAPI functions:

— **`BioAPI_Capture`**

— **`BioAPI_Process`**

— **`BioAPI_CreateTemplate`**

— **`BioAPI_VerifyMatch`**

— **`BioAPI_IdentifyMatch`**

— **`BioAPI_Capture`**

— **`BioAPI_Enroll`**

— **`BioAPI_Verify`**

— **`BioAPI_Identify`**

The application will receive GUI event notifications, which provide GUI-related information as a capture, enroll, verify, or identify operation proceeds. Table C.1 lists the GUI events that can occur.

Table C.1 — GUI events

GUI event	Explanation
select event	This GUI event is generated by a BSP before the start and after the end of a cycle of suboperations within a capture, verify, identify, or enroll operation. At the beginning of a cycle of suboperations, a GUI select event allows the application to specify, in its response to the notification callback, the subtype (see ISO/IEC 19785-1, 6.5.7) or subtypes of the sample that are to be captured (for example, left / right, index / point / middle / ring / little for finger) by the capture suboperation(s) within the cycle of suboperations that is about to begin. Some BSPs are able to perform simultaneous capture of multiple subtypes, or instances of a biometric modality (for example, all five fingers, or both irises) in a single capture suboperation, and therefore a bitmask is provided in the response to support multi-instance capture. At the end of a cycle of suboperations, a GUI select event informs the application about the completion of the cycle of suboperations (perhaps with some additional information), and allows the application to indicate whether the BSP should consider the entire operation as completed (with either **`BioAPI_OK`** or an error code returned to the application), cancel the operation (with an error code returned to the application), or discard the current results and start a new cycle of suboperations (without returning from the original BioSPI function call).
state event	This event is generated by a BSP before the start and after the end of each suboperation of an operation. The following five suboperations are specified: – capture: captures one or more instances of a modality using a sensor unit of the BSP and produces a sample of intermediate processed level, whose purpose can be verification, identification, or enrollment; – process: processes an intermediate sample with a purpose of either verification or identification and transforms it into a processed sample; – create template: processes an intermediate sample with a purpose of enrollment and transforms it into a reference template, possibly after choosing the "best" intermediate sample from a set; – verify match: performs a verification matching between a processed sample with a purpose of verification and a reference template; in the context of an enroll operation, this is also called a test verification; – identify match: performs an identification matching between a processed sample with a purpose of identification and multiple reference templates. The five suboperations have names similar to the names of five BioAPI functions because the task they perform is essentially the same, though at a different level of abstraction. For example, for **`BioSPI_Process`**, there is a single process suboperation, and likewise for **`BioSPI_CreateTemplate`** (a single create template suboperation), **`BioSPI_VerifyMatch`** (a single verify match suboperation), and **`BioSPI_IdentifyMatch`** (a single identify match suboperation). However, for the more-complex operations (**`BioSPI_Capture`**, **`BioSPI_Verify`**, **`BioSPI_Identify`**, and **`BioSPI_Enroll`**) there is a cycle of suboperations, consisting of one or more capture suboperations possibly followed by other suboperations. The whole cycle can be repeated one or more times without returning from the operation. If a BSP provides the enroll type Multiple-Capture to the **`BioAPI_GUI_SELECT_EVENT_HANDLER`** function, a suboperation cycle for **`BioSPI_Enroll`** consists of multiple capture suboperations followed by a create template suboperation. If a BSP provides the enroll type Test-Verify, a suboperation cycle for **`BioSPI_Enroll`** consists of a capture with a purpose of enrollment, a capture with the purpose of verification, a create template, a process, and a verify match (test verification).
progress event	The BSP can generate this event during any suboperation within a capture, process, create template, verify match, identify match, verify, identify, or enroll operation to inform the application about the progress of the suboperation. A GUI progress event notification for a capture suboperation can include streaming data (for example, a live video stream). Multiple GUI progress event notifications for a lengthy identify match suboperation can be used to inform the user about the percent of completion of the identify match.

The BSP is in control of a state machine associated with the operation, and provides state information, textual messages, and (where appropriate) bitmaps by calling a GUI event notification handler of the framework, which in turn calls back a GUI event notification handler of the application. After displaying the image to the subject (or to an operator controlling the biometric process), the application can determine whether to continue with the original function call, cancel it, or alter its course, by providing a response to the GUI event notification callback.

A GUI progress event can be used to provide streaming data to the application (intended for display to the subject or an operator) in the form of a series of bitmaps. During an identify match suboperation, a GUI progress event can be used to present the progress of the identification to the subject. Generation of the GUI progress event is a BSP option, and the BSP indicates in its options mask whether it implements this option.

The **`BioAPI_SubscribeToGUIEvents`** function subscribes the application to GUI select, state, and progress events, while the **`BioAPI_UnsubscribeFromGUIEvents`** function removes an existing GUI event subscription. All BSPs implementing the application-controlled GUI feature are required to support the corresponding BioSPI functions **`BioSPI_SubscribeToGUIEvents`** and **`BioSPI_UnsubscribeFromGUIEvents`**.

The interactions between the biometric application and a BSP supporting enrollment are determined by the enroll type provided by the BSP. Two enroll types are defined:

a) Test-Verify: During an enroll operation (invoked by a call to **`BioAPI_Enroll`**), the BSP that provides this enroll type performs:

1) a capture suboperation to produce a candidate for a reference template,

2) a second capture suboperation to obtain a test sample,

3) a create template suboperation (on the first sample),

4) a process suboperation (on the second sample), and

5) a verify match suboperation to determine if the test sample matches the candidate reference template.

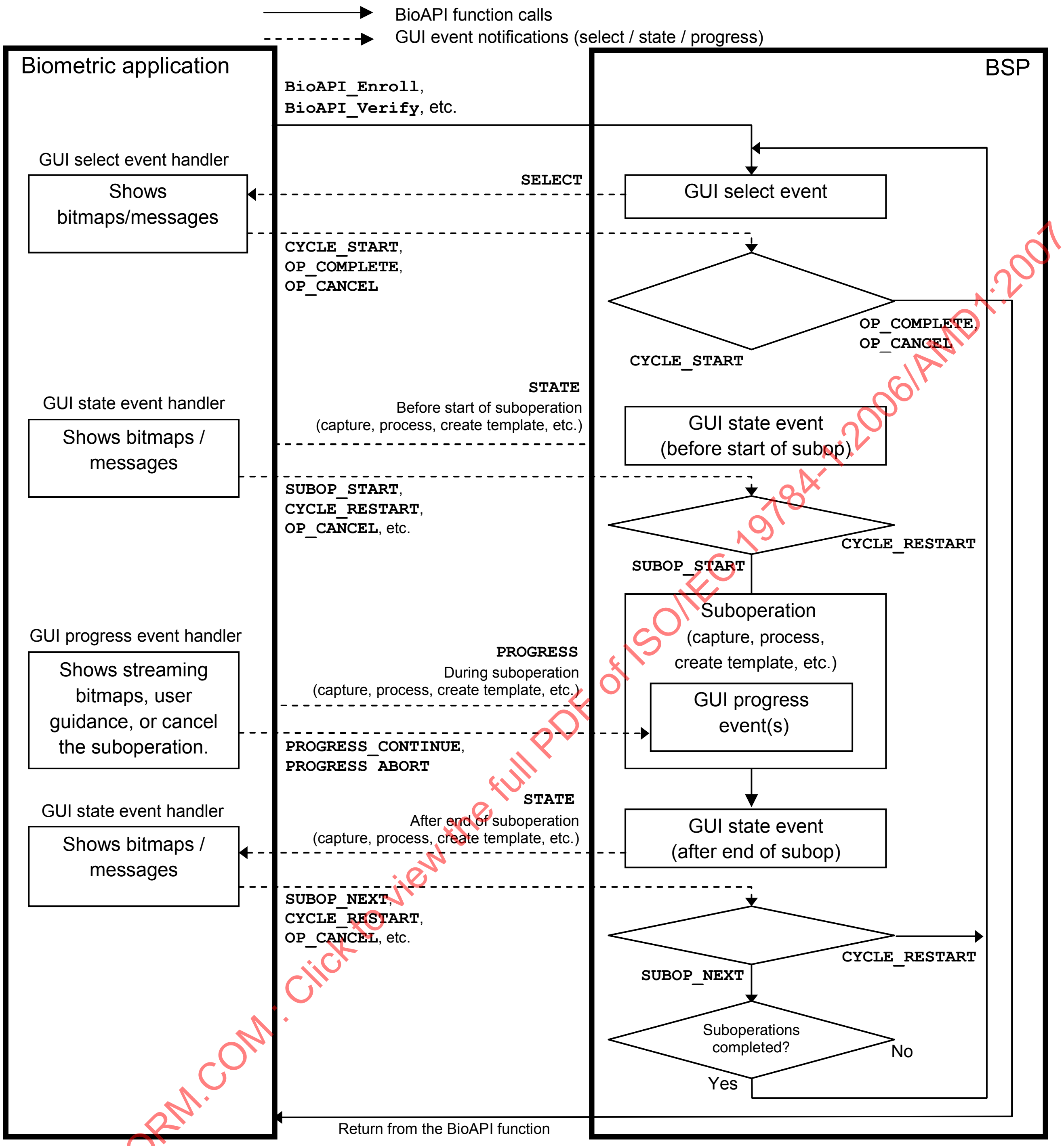
The candidate reference template is accepted if and only if the test verification is successful. Otherwise, the application can decide whether the cycle should be restarted or the enroll operation should fail.

b) Multiple-Capture: A BSP that provides this enroll type has certain established criteria for samples that are suitable for the production of a reference template. During an enroll operation, the BSP will perform one or more capture suboperations up to a specified limit (MaxNumEnrollInstances, see 7.60) in order to obtain a sample that satisfies its criteria.

NOTE: A BSP that performs (possibly) multiple captures to obtain a sample that can be used as a candidate for enrollment and then captures a further sample to match with the candidate (a combination of Test-Verify and Multiple-Capture) is not supported in BioAPI 2.0 or 2.1.

A BSP provides its enroll type as input to the BioAPI_GUI_SELECT_EVENT_HANDLER function, thus making it available to the application at the beginning of each enroll operation.

Figure C.4 shows a general flow of the implementation for the application-controlled GUI.


BioAPI function calls
GUI event notifications (select / state / progress)
Biometric application
BSP
BioAPI_Enroll, BioAPI_Verify, etc.
GUI select event handler
Shows bitmaps/messages
SELECT
GUI select event
CYCLE_START, OP_COMPLETE, OP_CANCEL
OP_COMPLETE, OP_CANCEL
CYCLE_START
GUI state event handler
Shows bitmaps / messages
STATE
Before start of suboperation (capture, process, create template, etc.)
GUI state event (before start of subop)
SUBOP_START, CYCLE_RESTART, OP_CANCEL, etc.
CYCLE_RESTART
SUBOP_START
Suboperation (capture, process, create template, etc.)
GUI progress event handler
Shows streaming bitmaps, user guidance, or cancel the suboperation.
PROGRESS
During suboperation (capture, process, create template, etc.)
GUI progress event(s)
PROGRESS_CONTINUE, PROGRESS_ABORT
GUI state event handler
Shows bitmaps / messages
STATE
After end of suboperation (capture, process, create template, etc.)
GUI state event (after end of subop)
SUBOP_NEXT, CYCLE_RESTART, OP_CANCEL, etc.
CYCLE_RESTART
SUBOP_NEXT
Suboperations completed?
No
Yes
Return from the BioAPI function

NOTE: In an actual implementation, there is a BioAPI Framework between an application and a BSP. The BioAPI Framework handles all BioAPI calls and event notifications.

Figure C.4 – An example of the general flow of a BioGUI implementation

C.7.2 Example of a sequence of GUI event notifications for an Enroll operation (enroll type = Test-Verify, no errors or restarts)

GUI event	Input parameters	Output parameters

select	Operation = **BioAPI_GUI_OPERATION_ENROLL**
	Moment = **BioAPI_GUI_MOMENT_BEFORE_START**
	ResultCode = n/a
	MaxNumEnrollSamples = 1
	SelectableInstances = left pointer finger, right pointer finger
	CapturedInstances = n/a
		Response = **BioAPI_GUI_RESPONSE_CYCLE_START**
		SelectedInstances = left pointer finger

State	Operation = **BioAPI_GUI_OPERATION_ENROLL**
	Suboperation = **BioAPI_GUI_SUBOPERATION_CAPTURE**
	Purpose = **BioAPI_PURPOSE_ENROLL**
	Moment = **BioAPI_GUI_MOMENT_BEFORE_START**
	ResultCode = n/a
	EnrollSampleIndex = 1
		Response = **BioAPI_GUI_RESPONSE_SUBOP_START**
		EnrollSampleIndexToRecapture = n/a

progress	Operation = **BioAPI_GUI_OPERATION_ENROLL**
	Suboperation = **BioAPI_GUI_SUBOPERATION_CAPTURE**
	Purpose = **BioAPI_PURPOSE_ENROLL**
	Moment = **BioAPI_GUI_MOMENT_DURING**
	SuboperationProgress = 20
		Response = **BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE**

progress	Operation = **BioAPI_GUI_OPERATION_ENROLL**
	Suboperation = **BioAPI_GUI_SUBOPERATION_CAPTURE**
	Purpose = **BioAPI_PURPOSE_ENROLL**
	Moment = **BioAPI_GUI_MOMENT_DURING**
	SuboperationProgress = 50
		Response = **BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE**

state	Operation = **BioAPI_GUI_OPERATION_ENROLL**
	Suboperation = **BioAPI_GUI_SUBOPERATION_CAPTURE**
	Purpose = **BioAPI_PURPOSE_ENROLL**
	Moment = **BioAPI_GUI_MOMENT_AFTER_END**
	ResultCode = **BioAPI_OK**
	EnrollSampleIndex = 1
		Response = **BioAPI_GUI_RESPONSE_SUBOP_NEXT**
		EnrollSampleIndexToRecapture = n/a

state	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = **`BioAPI_GUI_SUBOPERATION_CREATETEMPLATE`**
	Purpose = n/a
	Moment = **`BioAPI_GUI_MOMENT_BEFORE_START`**
	ResultCode = n/a
	EnrollSampleIndex = n/a
		Response = **`BioAPI_GUI_RESPONSE_SUBOP_START`**
		EnrollSampleIndexToRecapture = n/a

state	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_CREATETEMPLATE`
	Purpose = n/a
	Moment = **`BioAPI_GUI_MOMENT_AFTER_END`**
	ResultCode = **`BioAPI_OK`**
	EnrollSampleIndex = n/a
		Response = **`BioAPI_GUI_RESPONSE_SUBOP_NEXT`**
		EnrollSampleIndexToRecapture = n/a

state	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = **`BioAPI_GUI_SUBOPERATION_CAPTURE`**
	Purpose = **`BioAPI_PURPOSE_VERIFY`**
	Moment = **`BioAPI_GUI_MOMENT_BEFORE_START`**
	ResultCode = n/a
	EnrollSampleIndex = n/a
		Response = **`BioAPI_GUI_RESPONSE_SUBOP_START`**
		EnrollSampleIndexToRecapture = n/a

progress	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = **`BioAPI_GUI_SUBOPERATION_CAPTURE`**
	Purpose = **`BioAPI_PURPOSE_VERIFY`**
	Moment = **`BioAPI_GUI_MOMENT_DURING`**
	SuboperationProgress = 10
		Response = **`BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE`**

progress	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_CAPTURE`
	Purpose = `BioAPI_PURPOSE_VERIFY`
	Moment = `BioAPI_GUI_MOMENT_DURING`
	SuboperationProgress = 40
		Response = **`BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE`**

progress	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_CAPTURE`
	Purpose = `BioAPI_PURPOSE_VERIFY`
	Moment = `BioAPI_GUI_MOMENT_DURING`
	SuboperationProgress = 90
		Response = **`BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE`**

state	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_CAPTURE`
	Purpose = `BioAPI_PURPOSE_VERIFY`
	Moment = **`BioAPI_GUI_MOMENT_AFTER_END`**
	ResultCode = **`BioAPI_OK`**
	EnrollSampleIndex = n/a
		Response = **`BioAPI_GUI_RESPONSE_SUBOP_NEXT`**
		EnrollSampleIndexToRecapture = n/a

state	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = **`BioAPI_GUI_SUBOPERATION_PROCESS`**
	Purpose = n/a
	Moment = **`BioAPI_GUI_MOMENT_BEFORE_START`**
	ResultCode = n/a
	EnrollSampleIndex = n/a
		Response = **`BioAPI_GUI_RESPONSE_SUBOP_START`**
		EnrollSampleIndexToRecapture = n/a

state	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_PROCESS`
	Purpose = n/a
	Moment = **`BioAPI_GUI_MOMENT_AFTER_END`**
	ResultCode = **`BioAPI_OK`**
	EnrollSampleIndex = n/a
		Response = **`BioAPI_GUI_RESPONSE_SUBOP_NEXT`**
		EnrollSampleIndexToRecapture = n/a

state	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = **`BioAPI_GUI_SUBOPERATION_VERIFYMATCH`**
	Purpose = n/a
	Moment = **`BioAPI_GUI_MOMENT_BEFORE_START`**
	ResultCode = n/a
	EnrollSampleIndex = n/a
		Response = **`BioAPI_GUI_RESPONSE_SUBOP_START`**
		EnrollSampleIndexToRecapture = n/a

state	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_VERIFYMATCH`
	Purpose = n/a
	Moment = **`BioAPI_GUI_MOMENT_AFTER_END`**
	ResultCode = **`BioAPI_OK`**
	EnrollSampleIndex = n/a
		Response = **`BioAPI_GUI_RESPONSE_SUBOP_NEXT`**
		EnrollSampleIndexToRecapture = n/a

select	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Moment = **`BioAPI_GUI_MOMENT_AFTER_END`**
	ResultCode = **`BioAPI_OK`**
	MaxNumEnrollSamples = 1
	SelectableInstances = left pointer finger, right pointer finger
	CapturedInstances = left pointer finger
		Response = **`BioAPI_GUI_RESPONSE_OP_COMPLETE`**
		SelectedInstances = n/a

C.7.3 Example of a sequence of GUI event notifications for an Enroll operation (enroll type = Multiple-Capture, no errors or restarts, one recapture)

GUI event	Input parameters	Output parameters

select	Operation = **BioAPI_GUI_OPERATION_ENROLL**
	Moment = **BioAPI_GUI_MOMENT_BEFORE_START**
	ResultCode = n/a
	MaxNumEnrollSamples = 3
	SelectableInstances = left pointer finger, right pointer finger
	CapturedInstances = n/a
		Response = **BioAPI_GUI_RESPONSE_CYCLE_START**
		SelectedInstances = left pointer finger

state	Operation = **BioAPI_GUI_OPERATION_ENROLL**
	Suboperation = **BioAPI_GUI_SUBOPERATION_CAPTURE**
	Purpose = **BioAPI_PURPOSE_ENROLL**
	Moment = **BioAPI_GUI_MOMENT_BEFORE_START**
	ResultCode = n/a
	EnrollSampleIndex = 1
		Response = **BioAPI_GUI_RESPONSE_SUBOP_START**
		EnrollSampleIndexToRecapture = n/a

progress	Operation = **BioAPI_GUI_OPERATION_ENROLL**
	Suboperation = **BioAPI_GUI_SUBOPERATION_CAPTURE**
	Purpose = **BioAPI_PURPOSE_ENROLL**
	Moment = **BioAPI_GUI_MOMENT_DURING**
	SuboperationProgress = 18
		Response = **BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE**

progress	Operation = **BioAPI_GUI_OPERATION_ENROLL**
	Suboperation = **BioAPI_GUI_SUBOPERATION_CAPTURE**
	Purpose = **BioAPI_PURPOSE_ENROLL**
	Moment = **BioAPI_GUI_MOMENT_DURING**
	SuboperationProgress = 65
		Response = **BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE**

state	Operation = **BioAPI_GUI_OPERATION_ENROLL**
	Suboperation = **BioAPI_GUI_SUBOPERATION_CAPTURE**
	Purpose = **BioAPI_PURPOSE_ENROLL**
	Moment = **BioAPI_GUI_MOMENT_AFTER_END**
	ResultCode = **BioAPI_OK**
	EnrollSampleIndex = 1
		Response = **BioAPI_GUI_RESPONSE_SUBOP_NEXT**
		EnrollSampleIndexToRecapture = n/a

state	Operation = **BioAPI_GUI_OPERATION_ENROLL**
	Suboperation = **BioAPI_GUI_SUBOPERATION_CAPTURE**
	Purpose = **BioAPI_PURPOSE_ENROLL**
	Moment = **BioAPI_GUI_MOMENT_BEFORE_START**
	ResultCode = n/a
	EnrollSampleIndex = 2
		Response = **BioAPI_GUI_RESPONSE_SUBOP_START**
		EnrollSampleIndexToRecapture = n/a

progress	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_CAPTURE`
	Purpose = `BioAPI_PURPOSE_ENROLL`
	Moment = `BioAPI_GUI_MOMENT_DURING`
	SuboperationProgress = 30
		Response = `BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE`

progress	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_CAPTURE`
	Purpose = `BioAPI_PURPOSE_ENROLL`
	Moment = `BioAPI_GUI_MOMENT_DURING`
	SuboperationProgress = 80
		Response = `BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE`

progress	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_CAPTURE`
	Purpose = `BioAPI_PURPOSE_ENROLL`
	Moment = `BioAPI_GUI_MOMENT_DURING`
	SuboperationProgress = 95
		Response = `BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE`

state	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_CAPTURE`
	Purpose = `BioAPI_PURPOSE_ENROLL`
	Moment = `BioAPI_GUI_MOMENT_AFTER_END`
	ResultCode = `BioAPI_OK`
	EnrollSampleIndex = 2
		Response = `BioAPI_GUI_RESPONSE_SUBOP_NEXT`
		EnrollSampleIndexToRecapture = n/a

state	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_CAPTURE`
	Purpose = `BioAPI_PURPOSE_ENROLL`
	Moment = `BioAPI_GUI_MOMENT_BEFORE_START`
	ResultCode = n/a
	EnrollSampleIndex = 3
		Response = `BioAPI_GUI_RESPONSE_SUBOP_START`
		EnrollSampleIndexToRecapture = n/a

progress	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_CAPTURE`
	Purpose = `BioAPI_PURPOSE_ENROLL`
	Moment = `BioAPI_GUI_MOMENT_DURING`
	SuboperationProgress = 15
		Response = `BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE`

progress	Operation = `BioAPI_GUI_OPERATION_ENROLL`
	Suboperation = `BioAPI_GUI_SUBOPERATION_CAPTURE`
	Purpose = `BioAPI_PURPOSE_ENROLL`
	Moment = `BioAPI_GUI_MOMENT_DURING`
	SuboperationProgress = 70
		Response = `BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE`

progress	Operation = BioAPI_GUI_OPERATION_ENROLL
	Suboperation = BioAPI_GUI_SUBOPERATION_CAPTURE
	Purpose = BioAPI_PURPOSE_ENROLL
	Moment = BioAPI_GUI_MOMENT_DURING
	SuboperationProgress = 90
		Response = BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE

state	Operation = BioAPI_GUI_OPERATION_ENROLL
	Suboperation = BioAPI_GUI_SUBOPERATION_CAPTURE
	Purpose = BioAPI_PURPOSE_ENROLL
	Moment = BioAPI_GUI_MOMENT_AFTER_END
	ResultCode = BioAPI_OK
	EnrollSampleIndex = 3
		Response = BioAPI_GUI_RESPONSE_SUBOP_NEXT
		EnrollSampleIndexToRecapture = n/a

state	Operation = BioAPI_GUI_OPERATION_ENROLL
	Suboperation = BioAPI_GUI_SUBOPERATION_CREATETEMPLATE
	Purpose = n/a
	Moment = BioAPI_GUI_MOMENT_BEFORE_START
	ResultCode = n/a
	EnrollSampleIndex = n/a
		Response = BioAPI_GUI_RESPONSE_SUBOP_START
		EnrollSampleIndexToRecapture = n/a

state	Operation = BioAPI_GUI_OPERATION_ENROLL
	Suboperation = BioAPI_GUI_SUBOPERATION_CREATETEMPLATE
	Purpose = n/a
	Moment = BioAPI_GUI_MOMENT_AFTER_END
	ResultCode = BioAPIERR_LOW_QUALITY_REFERENCE_TEMPLATE
	EnrollSampleIndex = n/a
		Response = BioAPI_GUI_RESPONSE_RECAPTURE
		EnrollSampleIndexToRecapture = 2

state	Operation = BioAPI_GUI_OPERATION_ENROLL
	Suboperation = BioAPI_GUI_SUBOPERATION_CAPTURE
	Purpose = BioAPI_PURPOSE_ENROLL
	Moment = BioAPI_GUI_MOMENT_BEFORE_START
	ResultCode = n/a
	EnrollSampleIndex = 2
		Response = BioAPI_GUI_RESPONSE_SUBOP_START
		EnrollSampleIndexToRecapture = n/a

progress	Operation = BioAPI_GUI_OPERATION_ENROLL
	Suboperation = BioAPI_GUI_SUBOPERATION_CAPTURE
	Purpose = BioAPI_PURPOSE_ENROLL
	Moment = BioAPI_GUI_MOMENT_DURING
	SuboperationProgress = 45
		Response = BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE

state	Operation = BioAPI_GUI_OPERATION_ENROLL
	Suboperation = BioAPI_GUI_SUBOPERATION_CAPTURE
	Purpose = BioAPI_PURPOSE_ENROLL
	Moment = BioAPI_GUI_MOMENT_AFTER_END
	ResultCode = BioAPI_OK
	EnrollSampleIndex = 2
		Response = BioAPI_GUI_RESPONSE_NEXT_SUBOP
		EnrollSampleIndexToRecapture = n/a

state	Operation = BioAPI_GUI_OPERATION_ENROLL
	Suboperation = BioAPI_GUI_SUBOPERATION_CREATETEMPLATE
	Purpose = n/a
	Moment = BioAPI_GUI_MOMENT_BEFORE_START
	ResultCode = n/a
	EnrollSampleIndex = n/a
		Response = BioAPI_GUI_RESPONSE_START_SUBOP
		EnrollSampleIndexToRecapture = n/a

state	Operation = BioAPI_GUI_OPERATION_ENROLL
	Suboperation = BioAPI_GUI_SUBOPERATION_CREATETEMPLATE
	Purpose = n/a
	Moment = BioAPI_GUI_MOMENT_AFTER_END
	ResultCode = BioAPI_OK
	EnrollSampleIndex = n/a
		Response = BioAPI_GUI_RESPONSE_NEXT_SUBOP
		EnrollSampleIndexToRecapture = n/a

select	Operation = BioAPI_GUI_OPERATION_ENROLL
	Moment = BioAPI_GUI_MOMENT_AFTER_END
	ResultCode = BioAPI_OK
	MaxNumEnrollSamples = 1
	SelectableInstances = left pointer finger, right pointer finger
	CapturedInstances = left pointer finger
		Response = BioAPI_GUI_RESPONSE_OP_COMPLETE
		SelectedInstances = n/a

select	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Moment = BioAPI_GUI_MOMENT_BEFORE_START
	ResultCode = n/a
	MaxNumEnrollSamples = n/a
	SelectableInstances = right
	CapturedInstances = n/a
		Response = BioAPI_GUI_RESPONSE_START_CYCLE
		SelectedInstances = right

C.7.4 Example of a sequence of GUI event notifications for an Identify operation (no errors or restarts)

GUI event	Input parameters	Output parameters

state	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Suboperation = BioAPI_GUI_SUBOPERATION_CAPTURE
	Purpose = BioAPI_PURPOSE_IDENTIFY
	Moment = BioAPI_GUI_MOMENT_BEFORE_START
	ResultCode = n/a
	EnrollSampleIndex = n/a
		Response = BioAPI_GUI_RESPONSE_START_SUBOP
		EnrollSampleIndexToRecapture = n/a

progress	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Suboperation = BioAPI_GUI_SUBOPERATION_CAPTURE
	Purpose = BioAPI_PURPOSE_IDENTIFY
	Moment = BioAPI_GUI_MOMENT_DURING
	SuboperationProgress = 32
		Response = BioAPI_GUI_RESPONSE_CONTINUE_SUBOP

progress	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Suboperation = BioAPI_GUI_SUBOPERATION_CAPTURE
	Purpose = BioAPI_PURPOSE_IDENTIFY
	Moment = BioAPI_GUI_MOMENT_DURING
	SuboperationProgress = 70
		Response = BioAPI_GUI_RESPONSE_CONTINUE_SUBOP

state	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Suboperation = BioAPI_GUI_SUBOPERATION_CAPTURE
	Purpose = BioAPI_PURPOSE_IDENTIFY
	Moment = BioAPI_GUI_MOMENT_AFTER_END
	ResultCode = BioAPI_OK
	EnrollSampleIndex = 1
		Response = BioAPI_GUI_RESPONSE_NEXT_SUBOP
		EnrollSampleIndexToRecapture = n/a

state	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Suboperation = BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH
	Purpose = n/a
	Moment = BioAPI_GUI_MOMENT_BEFORE_START
	ResultCode = n/a
	EnrollSampleIndex = n/a
		Response = BioAPI_GUI_RESPONSE_START_SUBOP
		EnrollSampleIndexToRecapture = n/a

progress	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Suboperation = BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH
	Purpose = n/a
	Moment = BioAPI_GUI_MOMENT_DURING
	SuboperationProgress = 9
		Response = BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE

progress	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Suboperation = BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH
	Purpose = n/a
	Moment = BioAPI_GUI_MOMENT_DURING
	SuboperationProgress = 30
		Response = BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE

progress	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Suboperation = BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH
	Purpose = n/a
	Moment = BioAPI_GUI_MOMENT_DURING
	SuboperationProgress = 64
		Response = BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE

progress	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Suboperation = BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH
	Purpose = n/a
	Moment = BioAPI_GUI_MOMENT_DURING
	SuboperationProgress = 83
		Response = BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE

progress	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Suboperation = BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH
	Purpose = n/a
	Moment = BioAPI_GUI_MOMENT_DURING
	SuboperationProgress = 98
		Response = BioAPI_GUI_RESPONSE_PROGRESS_CONTINUE

state	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Suboperation = BioAPI_GUI_SUBOPERATION_IDENTIFYMATCH
	Purpose = n/a
	Moment = **BioAPI_GUI_MOMENT_AFTER_END**
	ResultCode = **BioAPI_OK**
	EnrollSampleIndex = n/a
		Response = **BioAPI_GUI_RESPONSE_SUBOP_NEXT**
		EnrollSampleIndexToRecapture = n/a

select	Operation = BioAPI_GUI_OPERATION_IDENTIFY
	Moment = **BioAPI_GUI_MOMENT_AFTER_END**
	ResultCode = **BioAPI_OK**
	MaxNumEnrollSamples = n/a
	SelectableInstances = right
	CapturedInstances = right
		Response = **BioAPI_GUI_RESPONSE_OP_COMPLETE**
		SelectedInstances = n/a

2-44) Replace C.8.4 with the following text:

C.8.4 There are a variety of ways in which the biometric and smartcard components (software and hardware) can interact to perform the MOC operation. Figure C.3 shows one high-level process flow for such an operation.

2-45) Add the following text as a new subclause D.3:

D.3 Calling sequences with BioGUI (BioAPI 2.1)

The examples in this subclause apply only when the BioAPI version number in use is 2.1.

D.3.1 Enroll with BioGUI

If a BSP supports the function **`BioAPI_SubscribeToGUIEvents`** and the enroll type provided by the BSP is Test-Verify, a possible call sequence at enrollment is as follows.

```
int ApplicationControlledEnrollFunction()
{
   BioAPI_RETURN          bioReturn;
   BioAPI_BIR_HANDLE      EnrolledTemplate;
   BioAPI_HANDLE          BSPHandle;
   BioAPI_GUI_SELECT_EVENT_HANDLER      EnrollGUISelectEventHandler;
   BioAPI_GUI_STATE_EVENT_HANDLER    EnrollGUIStateEventHandler;
   BioAPI_GUI_PROGRESS_EVENT_HANDLER EnrollGUIProgressEventHandler;


   // Set GUI event handlers.
   bioReturn = BioAPI_SubscribeToGUIEvents(
                 NULL,
                 NULL,
                 &BSPHandle,
                 EnrollGUISelectEventHandler,
                 NULL,
                 EnrollGUIStateEventHandler,
                 NULL,
                 EnrollGUIProgressEventHandler,
                 NULL);

   if(bioReturn != BioAPI_OK)
   {
      printf("BioAPI Error Code:       %d\n", bioReturn);
       return 0;
   }

   // When the BioAPI_Enroll function is called, the GUI Events
   // will be called before this function returns.
   bioReturn = BioAPI_Enroll(
                 BSPHandle,
                 BioAPI_PURPOSE_ENROLL
                 NULL,
                 &EnrolledTemplate,
                 NULL,-1, NULL, NULL);

   // After the BioAPI_Enroll function returns, the application can
   // clear the callback entries by calling the
   // BioAPI_UnsubscribeFromGUIEvents function.
```

```
    BioAPI_UnsubscribeFromGUIEvents (
                  NULL,
                  NULL,
                  &BSPHandle,
                  EnrollGUISelectEventHandler,
                  NULL,
                  EnrollGUIStateEventHandler,
                  NULL,
                  EnrollGUIProgressEventHandler,
                  NULL);
    if(bioReturn != BioAPI_OK)
    {
        printf("BioAPI Error Code:        %d\n", bioReturn);
       return 0;
    }

    return 1;
}

BIOAPI_RETURN CALLBACK EnrollGUISelectEventHandler
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    uint32_t  EnrollType,
    const void *GUISelectEventHandlerCtx,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
    int32_t MaxNumEnrollSamples,
    BioAPI_BIR_SUBTYPE_MASK SelectableInstances,
    BioAPI_BIR_SUBTYPE_MASK *SelectedInstances,
    BioAPI_BIR_SUBTYPE_MASK CapturedInstances,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response)
{
// Show a GUI to the subject for selecting a specific biometric
// subtype to be enrolled.
// Set the SelectedInstances parameter based on the subtype(s)
// selected by the subject.
// If this is a GUI select event generated before a suboperation
// cycle (Moment = BioAPI_GUI_MOMENT_BEFORE_START), then set
// the Response parameter to BioAPI_GUI_RESPONSE_CYCLE_START
// if the subject wishes to proceed with the enrollment, or
// to BioAPI_GUI_RESPONSE_OP_CANCEL if the subject wishes to
// cancel the enrollment.
// If this is a GUI select event generated after a suboperation
// cycle (Moment = BioAPI_GUI_MOMENT_AFTER_END), then set
// the Response parameter to BioAPI_GUI_RESPONSE_OP_COMPLETE
// if the subject is satisfied with the enrollment,
// or to BioAPI_GUI_RESPONSE_CYCLE_RESTART if the subject
// wishes the cycle to be repeated, or to
// BioAPI_GUI_RESPONSE_OP_CANCEL if the subject wishes
// to cancel the enrollment.

    return BioAPI_OK ;
}

BioAPI_RETURN CALLBACK EnrollGUIStateEventHandler
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
        const void *GUIStateEventHandlerCtx,
        BioAPI_GUI_OPERATION Operation,
        BioAPI_GUI_SUBOPERATION Suboperation,
        BioAPI_BIR_PURPOSE Purpose,
        BioAPI_GUI_MOMENT Moment,
        BioAPI_RETURN ResultCode,
        int32_t  EnrollSampleIndex,
        const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
        const uint8_t  *Text,
        BioAPI_GUI_RESPONSE  *Response,
        int32_t  *EnrollSampleIndexToRecapture)
     {
     // Show a GUI based on the values of the parameters
     // Operation, Suboperation, Purpose, Moment, and
     // ResultCode provided by the BSP. If the Text is
     // provided, the text can be displayed.

        switch(Suboperation){
        case BioAPI_GUI_SUBOPERATION_CAPTURE:

           switch(Purpose){
           case BioAPI_PURPOSE_ENROLL:
           case BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY:
           case BioAPI_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY:

              switch(Moment){
              case BioAPI_GUI_MOMENT_BEFORE_START:
                 // Message example: "Click [Capture] button to
                 // start image acquisition."
                 break;
              case BioAPI_GUI_MOMENT_AFTER_END:
                 // Message example 1: "Capture has finished successfully."
                 // Message example 2: "Capture failure."
                 break;
              }
              break;

           case BioAPI_PURPOSE_VERIFY:

              switch(Moment){
              case BioAPI_GUI_MOMENT_BEFORE_START:
                 // Message example: "Click [Capture] to start image
                 // acquisition for test verification."
                 break;
              case BioAPI_GUI_MOMENT_AFTER_END:
                 // Message example 1: "Capture has finished successfully."
                 // Message example 2: "Capture failure."
                 break;
              }
              break;

           }
           break;

        // other suboperations
        }

        return BioAPI_OK ;
     }
```

```
BioAPI_RETURN CALLBACK EnrollGUIProgressEventHandler
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    const void *GUIProgressEventHandlerCtx,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_SUBOPERATION Suboperation,
    BioAPI_BIR_PURPOSE Purpose,
    BioAPI_GUI_MOMENT Moment,
    uint8_t SuboperationProgress,
    const BioAPI_GUI_BITMAP_ARRAY *Bitmaps,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response)
{
    // Show streaming data to screen.
}
```

Once the template is returned, it can be placed in a data store for later retrieval and verification (see D.2).

D.3.2 Perform a verification with BioGUI

If a BSP supports the **BioAPI_SubscribeToGUIEvents** function, a possible calling sequence at verification using **BioAPI_Verify** is as follows.

```
int ApplicationControlledVerifyFunction ()
{
    BioAPI_INPUT_BIR birEnroll;
    BioAPI_BIR_HEADER birHeader;
    int MaxFMR, AchievedFMR;
    BioAPI_BOOL bPrecedence, bResponse;

    BioAPI_HANDLE    BSPHandle;
    BioAPI_GUI_SELECT_EVENT_HANDLER      VerifyGUISelectEventHandler;
    BioAPI_GUI_STATE_EVENT_HANDLER    VerifyGUIStateEventHandler;
    BioAPI_GUI_PROGRESS_EVENT_HANDLER VerifyGUIProgressEventHandler;

    // Set GUI Events.
    bioReturn = BioAPI_SubscribeToGUIEvents(
                 NULL,
                 NULL,
                 &BSPHandle,
                 VerifyGUISelectEventHandler,
                 NULL,
                 VerifyGUIStateEventHandler,
                 NULL,
                 VerifyGUIProgressEventHandler,
                 NULL);

    // When the BioAPI_Verify function is called, the GUI
    // state callback will be called before this function returns.

    MaxFMR = 1;

    bioReturn = BioAPI_Verify( BSPHandle,
                        &MaxFMR,
                        NULL,
                        &bPrecedence,
                        &birEnroll,
                        NULL,
                        &bResponse,
                          &AchievedFMR,
                          NULL,
                          NULL,
                          -1,
                          NULL);

    free(birEnroll.InputBIR.BIR);

    // After the BioAPI_Verify function returns, the application
    // can clear the callback entries by calling
    // the BioAPI_UnsubscribeFromGUIEvents function.
    BioAPI_UnsubscribeFromGUIEvents (
                    NULL,
                    NULL,
                    &BSPHandle,
                    VerifyGUISelectEventHandler,
                    NULL,
                    VerifyGUIStateEventHandler,
                    NULL,
                    VerifyGUIProgressEventHandler,
                    NULL);

    if(bioReturn != BioAPI_OK)
    {
        printf("BioAPI Error Code:       %d\n", bioReturn);
        return 0;
    }
    else if(bResponse != BioAPI_TRUE)
    {
        printf("No Match\n");
        return 0;
    }
    else
    {
        printf("Match\n");
        return 1;
    }
}

BIOAPI_RETURN CALLBACK VerifyGUISelectEventHandler
    (const BioAPI_UUID *BSPUuid,
    BioAPI_UNIT_ID UnitID,
    const BioAPI_HANDLE *BSPHandle,
    uint32_t  EnrollType,
    const void *GUISelectEventHandlerCtx,
    BioAPI_GUI_OPERATION Operation,
    BioAPI_GUI_MOMENT Moment,
    BioAPI_RETURN ResultCode,
    int32_t MaxNumEnrollSamples,
    BioAPI_BIR_SUBTYPE_MASK SelectableInstances,
    BioAPI_BIR_SUBTYPE_MASK *SelectedInstances,
    BioAPI_BIR_SUBTYPE_MASK CapturedInstances,
    const uint8_t *Text,
    BioAPI_GUI_RESPONSE *Response)
{
// Show a GUI to the subject for selecting a specific
// biometric subtype to be verified.
// Set the SelectedInstances parameter based on the subtype(s)
// selected by the subject.
// If this is a GUI select event generated before a suboperation
```