CEN

WORKSHOP

AGREEMENT

CWA 16926-78

January 2023

ICS 35.240.40; 35.200; 35.240.15

English version

Extensions for Financial Services (XFS) interface specification Release 3.50 - Part 78: Biometrics Device Class Interface Proposal - Programmer's Reference - Migration from Version 3.40 (CWA 16926:2020) to Version 3.50 (this CWA)

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

Table of Contents

European Foreword

This CEN Workshop Agreement has been developed in accordance with the CEN-CENELEC Guide 29 “CEN/CENELEC Workshop Agreements – The way to rapid consensus” and with the relevant provisions of CEN/CENELEC Internal Regulations – Part 2. It was approved by a Workshop of representatives of interested parties on 2022-11-08, the constitution of which was supported by CEN following several public calls for participation, the first of which was made on 1998-06-24. However, this CEN Workshop Agreement does not necessarily include all relevant stakeholders.

The final text of this CEN Workshop Agreement was provided to CEN for publication on 2022-11-18.

The following organizations and individuals developed and approved this CEN Workshop Agreement:

- AURIGA SPA
- CIMA SPA
- DIEBOLD NIXDORF SYSTEMS GMBH
- FIS BANKING SOLUTIONS UK LTD (OTS)
- FUJITSU TECHNOLOGY SOLUTIONS
- GLORY LTD
- GRG BANKING EQUIPMENT HK CO LTD
- HITACHI CHANNEL SOLUTIONS CORP
- HYOSUNG TNS INC
- JIANGSU GUOGUANG ELECTRONIC INFORMATION TECHNOLOGY
- KAL
- KEBA HANDOVER AUTOMATION GMBH
- NCR FSG
- NEXUS SOFTWARE
- OBERTHUR CASH PROTECTION
- OKI ELECTRIC INDUSTRY SHENZHEN
- SALZBURGER BANKEN SOFTWARE
- SECURE INNOVATION
- SIGMA SPA

It is possible that some elements of this CEN/CWA may be subject to patent rights. The CEN-CENELEC policy on patent rights is set out in CEN-CENELEC Guide 8 “Guidelines for Implementation of the Common IPR Policy on Patents (and other statutory intellectual property rights based on inventions)”. CEN shall not be held responsible for identifying any or all such patent rights.

The Workshop participants have made every effort to ensure the reliability and accuracy of the technical and non-technical content of CWA 16926-19, but this does not guarantee, either explicitly or implicitly, its correctness. Users of CWA 16926-19 should be aware that neither the Workshop participants, nor CEN can be held liable for damages or losses of any kind whatsoever which may arise from its application. Users of CWA 16926-19 do so on their own responsibility and at their own risk.

The CWA is published as a multi-part document, consisting of:

Part 1: Application Programming Interface (API) - Service Provider Interface (SPI) - Programmer's Reference

Part 2: Service Classes Definition - Programmer's Reference

Part 3: Printer and Scanning Device Class Interface - Programmer's Reference

Part 4: Identification Card Device Class Interface - Programmer's Reference

Part 5: Cash Dispenser Device Class Interface - Programmer's Reference

Part 6: PIN Keypad Device Class Interface - Programmer's Reference

Part 7: Check Reader/Scanner Device Class Interface - Programmer's Reference

Part 8: Depository Device Class Interface - Programmer's Reference

Part 9: Text Terminal Unit Device Class Interface - Programmer's Reference

Part 10: Sensors and Indicators Unit Device Class Interface - Programmer's Reference

Part 11: Vendor Dependent Mode Device Class Interface - Programmer's Reference

Part 12: Camera Device Class Interface - Programmer's Reference

Part 13: Alarm Device Class Interface - Programmer's Reference

Part 14: Card Embossing Unit Device Class Interface - Programmer's Reference

Part 15: Cash-In Module Device Class Interface - Programmer's Reference

Part 16: Card Dispenser Device Class Interface - Programmer's Reference

Part 17: Barcode Reader Device Class Interface - Programmer's Reference

Part 18: Item Processing Module Device Class Interface - Programmer's Reference

Part 19: Biometrics Device Class Interface - Programmer's Reference

Parts 20 - 28: Reserved for future use.

Parts 29 through 47 constitute an optional addendum to this CWA. They define the integration between the SNMP standard and the set of status and statistical information exported by the Service Providers.

Part 29: XFS MIB Architecture and SNMP Extensions - Programmer’s Reference

Part 30: XFS MIB Device Specific Definitions - Printer Device Class

Part 31: XFS MIB Device Specific Definitions - Identification Card Device Class

Part 32: XFS MIB Device Specific Definitions - Cash Dispenser Device Class

Part 33: XFS MIB Device Specific Definitions - PIN Keypad Device Class

Part 34: XFS MIB Device Specific Definitions - Check Reader/Scanner Device Class

Part 35: XFS MIB Device Specific Definitions - Depository Device Class

Part 36: XFS MIB Device Specific Definitions - Text Terminal Unit Device Class

Part 37: XFS MIB Device Specific Definitions - Sensors and Indicators Unit Device Class

Part 38: XFS MIB Device Specific Definitions - Camera Device Class

Part 39: XFS MIB Device Specific Definitions - Alarm Device Class

Part 40: XFS MIB Device Specific Definitions - Card Embossing Unit Class

Part 41: XFS MIB Device Specific Definitions - Cash-In Module Device Class

Part 42: Reserved for future use.

Part 43: XFS MIB Device Specific Definitions - Vendor Dependent Mode Device Class

Part 44: XFS MIB Application Management

Part 45: XFS MIB Device Specific Definitions - Card Dispenser Device Class

Part 46: XFS MIB Device Specific Definitions - Barcode Reader Device Class

Part 47: XFS MIB Device Specific Definitions - Item Processing Module Device Class

Part 48: XFS MIB Device Specific Definitions - Biometrics Device Class

Parts 49 - 60 are reserved for future use.

Part 61: Application Programming Interface (API) - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Service Provider Interface (SPI) - Programmer's Reference

Part 62: Printer and Scanning Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version

3.50 (this CWA) - Programmer's Reference

Part 63: Identification Card Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 64: Cash Dispenser Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 65: PIN Keypad Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 66: Check Reader/Scanner Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 67: Depository Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 68: Text Terminal Unit Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 69: Sensors and Indicators Unit Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 70: Vendor Dependent Mode Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 71: Camera Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 72: Alarm Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 73: Card Embossing Unit Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 74: Cash-In Module Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 75: Card Dispenser Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 76: Barcode Reader Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 77: Item Processing Module Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

Part 78: Biometric Device Class Interface - Migration from Version 3.40 (CWA 16296:2020) to Version 3.50 (this CWA) - Programmer's Reference

In addition to these Programmer's Reference specifications, the reader of this CWA is also referred to a complementary document, called Release Notes. The Release Notes contain clarifications and explanations on the CWA specifications, which are not requiring functional changes. The current version of the Release Notes is available online from: https://www.cencenelec.eu/areas-of-work/cen-sectors/digital-society-cen/cwa-download-area/.

The information in this document represents the Workshop's current views on the issues discussed as of the date of publication. It is provided for informational purposes only and is subject to change without notice. CEN makes no warranty, express or implied, with respect to this document.

Revision History:

3.40	December 06, 2019	Initial Release.
3.50	November 18, 2022	For a description of changes from version 3.40 to version 3.50 see the BIO 3.50 Migration document.

1. Introduction

1.1 Background to Release 3.50

The CEN/XFS Workshop aims to promote a clear and unambiguous specification defining a multi-vendor software interface to financial peripheral devices. The XFS (eXtensions for Financial Services) specifications are developed within the CEN (European Committee for Standardization/Information Society Standardization System) Workshop environment. CEN Workshops aim to arrive at a European consensus on an issue that can be published as a CEN Workshop Agreement (CWA).

The CEN/XFS Workshop encourages the participation of both banks and vendors in the deliberations required to create an industry standard. The CEN/XFS Workshop achieves its goals by focused sub-groups working electronically and meeting quarterly.

Release 3.50 of the XFS specification is based on a C API and is delivered with the continued promise for the protection of technical investment for existing applications. This release of the specification extends the functionality and capabilities of the existing devices covered by the specification:

- Addition of E2E security
- PIN Password Entry

1.2 XFS Service-Specific Programming

The service classes are defined by their service-specific commands and the associated data structures, error codes, messages, etc. These commands are used to request functions that are specific to one or more classes of Service Providers, but not all of them, and therefore are not included in the common API for basic or administration functions.

When a service-specific command is common among two or more classes of Service Providers, the syntax of the command is as similar as possible across all services, since a major objective of XFS is to standardize function codes and structures for the broadest variety of services. For example, using the **WFSExecute** function, the commands to read data from various services are as similar as possible to each other in their syntax and data structures.

In general, the specific command set for a service class is defined as a superset of the specific capabilities likely to be provided by the developers of the services of that class; thus any particular device will normally support only a subset of the defined command set.

There are three cases in which a Service Provider may receive a service-specific command that it does not support:

The requested capability is defined for the class of Service Providers by the XFS specification, the particular vendor implementation of that service does not support it, and the unsupported capability is ***not*** considered to be fundamental to the service. In this case, the Service Provider returns a successful completion, but does no operation. An example would be a request from an application to turn on a control indicator on a passbook printer; the Service Provider recognizes the command, but since the passbook printer it is managing does not include that indicator, the Service Provider does no operation and returns a successful completion to the application.

The requested capability is defined for the class of Service Providers by the XFS specification, the particular vendor implementation of that service does not support it, and the unsupported capability ***is*** considered to be fundamental to the service. In this case, a WFS_ERR_UNSUPP_COMMAND error for Execute commands or WFS_ERR_UNSUPP_CATEGORY error for Info commands is returned to the calling application. An example would be a request from an application to a cash dispenser to retract items where the dispenser hardware does not have that capability; the Service Provider recognizes the command but, since the cash dispenser it is managing is unable to fulfil the request, returns this error.

The requested capability is ***not*** defined for the class of Service Providers by the XFS specification. In this case, a WFS_ERR_INVALID_COMMAND error for Execute commands or WFS_ERR_INVALID_CATEGORY error for Info commands is returned to the calling application.

This design allows implementation of applications that can be used with a range of services that provide differing subsets of the functionalities that are defined for their service class. Applications may use the **WFSGetInfo** and **WFSAsyncGetInfo** commands to inquire about the capabilities of the service they are about to use, and modify their behavior accordingly, or they may use functions and then deal with error returns to make decisions as to how to use the service.

2. Biometric Devices

Biometrics refers to metrics related to human characteristics and biology. Biometrics authentication can be used as a form of identification and/or access control. This is an overview of biometrics, as well as an introduction to the terminology used in this document. It introduces to XFS the concept of scanning a person's biometric data in raw image form (raw biometric data), then processing it into a smaller more concise form that is easier to manage (biometric template data). The first scan of a user is called **ENROLLMENT** as the user is effectively being enrolled into a scheme by recording their biometric data. Thereafter subsequent scans of the user can be compared to the original data in order to verify who they say they are (**VERIFICATION**), or alternatively used to identify them as a specific individual (**IDENTIFICATION**). These concepts are explained below in more detail.

2.1 Enrollment

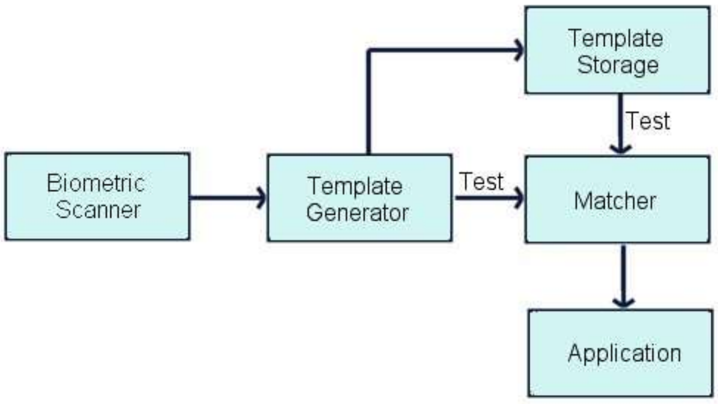
The first time an individual uses a biometric device it is called Enrollment. During enrollment, biometric data from an individual is captured and stored somewhere, for example on a smart card or in a server/host database. Normally the raw biometric data captured will be processed and converted to a smaller format that is used for subsequent comparison. This format is referred to in this document as a template. A template is a synthesis of the relevant characteristics extracted from the original raw data. Elements of the biometric data that are not used in the matching algorithm are discarded in the template to reduce the file size and to protect the identity of the enrollee.

2.2 Biometric Matching

During the matching phase, the obtained template is passed to a matcher which compares it to other existing templates and a probable match is calculated, either as a Boolean true or false or as a threshold indicating the likelihood of a match. With regard to matching, biometric systems commonly have two different basic modes of operation: Verification and Identification:

Verification: performs a one-to-one comparison of captured biometric data with a specific template in order to verify that an individual is the person they claim to be.

Identification: the system performs a one-to-many comparison of captured biometric data in order to establish a person's identity.



Note: The above diagram does not make any assumptions about where the actual matching takes place. The interface provided is versatile enough to be able to support three basic Biometric systems:

Match on server: The biometric template data is stored on a server or host. When scanning takes place biometric data is sent to the server, which does the actual identification or verification.

Match on card: The biometric enrollment data for an individual is stored on a smart card/personal device. The device scans a user then returns the biometric template information to the application. This data is then sent to the card, and an application on the smart card chip does the comparison, returning the result to the application.

Match on device: The biometric enrollment data for an individual is stored on a smart card or host. The enrollment data is read from the card or host and into the device, which then compares it to scanned information, returning the result to the application.

2.3 Biometric Device Types

There are many different varieties of biometric hardware, this XFS biometrics specification supports three main different types of device:

1. **Devices which only support scanning and returning biometric data**
   In this case the device is a simple biometric scanning device, User data is scanned using the WFS_CMD_BIO_READ command, but matching is performed externally, for example on a smart card or on a server. In this case the WFS_CMD_BIO_MATCH and WFS_CMD_BIO_SET_MATCH commands are not supported.
2. **Devices which support a separate scan and match functionality**
   These devices scan and perform a comparison as separate operations. Existing biometric data is first imported using the WFS_CMD_BIO_IMPORT command. When the WFS_CMD_BIO_READ command is then called the scanned user data is temporarily stored. The WFS_CMD_BIO_MATCH command is then called to perform the comparison and return the result.
3. **Devices which support a combined scan and match functionality**
   These devices scan and perform a comparison as a single operation. Existing biometric data is first imported using the WFS_CMD_BIO_IMPORT command. In this case the WFS_CMD_BIO_SET_MATCH command must be called first, either as a one~~–~~-time call or before each WFS_CMD_BIO_READ command. The purpose of the WFS_CMD_BIO_SET_MATCH command is to set the criteria for matching. When the WFS_CMD_BIO_READ command is then called it scans the user's biometric data and also performs the comparison as a single operation. The WFS_CMD_BIO_MATCH command is then called to return the result of the comparison.

2.4 Biometric Data Security

It is recommended that biometric data should be treated with the same strict caution as any other identifying and sensitive information. A well designed biometric data handling architecture should always be designed to protect against internal tampering, external attacks and other malicious threats. There are various ways of implementing good security of which three are listed below:

- **Multi Modal Biometrics**
  A Uni-Modal biometric system relies on data taken from a single source of information for authentication, for example a single fingerprint reading device. In contrast, Multi-Modal biometric systems work on the premise that it is more secure to accept information from two or more biometric inputs. As an example a user could provide a fingerprint in addition to facial recognition, a positive match from two physical characteristics improves the chances of a positive identification and mitigates the possibility that biometric data has been cloned.
- **Data Encryption**
  Biometric data should be encrypted where possible. The BIO specification provides for this by allowing an encryption key to be specified whenever data is exchanged between an application and a BIO Service Provider. In addition, the key management interface methods of the PIN device class can be used for key management. This can be done by using the standard XFS compound device mechanism to implement a BIO Service provider as a compound device together with a PIN device class Service Provider. The device compounding mechanism is described in the XFS API specification. In this case the BIO Service Provider would implement the biometric methods necessary to read and return data, while the key loading, reporting etc~~,~~. functions of the PIN Service Provider interface would be implemented in order to provide key management.

3. References

1. XFS Application Programming Interface (API)/Service Provider Interface (~~-~~SPI), Programmer's Reference, Revision 3.~~40~~50
2. ANSI INCITS 381-2004 Information Technology - Finger Image-Based Data Interchange Format~~.~~
3. ANSI INCITS 378-2004 Information Technology - Finger Minutiae Format for Data Interchange~~.~~
4. ISO/IEC 19794-4:2005 Information technology - Biometric data interchange formats - Part 4: Finger image data~~.~~
5. ISO/IEC 19794-2:2005 Information technology - Biometric data interchange formats - Part 2: Finger minutiae data~~.~~

4. Info Commands

4.1 WFS_INF_BIO_STATUS

Description This command is used to obtain the status of the biometric device. It may also return vendor-specific status information.

Input Param None.

Output Param LPWFSBIOSTATUS lpStatus;

```
typedef struct _wfs_bio_status
     {
     WORD                 fwDevice;
     DWORD                dwSubject;
     BOOL                 bCaptured;
     DWORD                dwDataPersistence;
     DWORD                dwRemainingStorage;
     LPSTR                lpszExtra;
     WORD                 wDevicePosition;
     DWORD                dwGuidLights[WFS_BIO_GUIDLIGHTS_SIZE];
     USHORT               usPowerSaveRecoveryTime;
     WORD                 wAntiFraudModule;
     } WFSBIOSTATUS, *LPWFSBIOSTATUS;
```

fwDevice
Specifies the state of the biometric device as one of the following values:

Value	Meaning
WFS_BIO_DEVONLINE	The device is present, powered on and online (i.e. operational, not busy processing a request and not in an error state).
WFS_BIO_DEVOFFLINE	The device is offline (e.g. the operator has taken the device offline by turning a switch).
WFS_BIO_DEVPOWEROFF	The device is powered off or physically not connected.
WFS_BIO_DEVNODEVICE	There is no device intended to be there; e.g. this type of self-service machine does not contain such a device or it is internally not configured.
WFS_BIO_DEVHWERROR	The device is present but inoperable due to a hardware fault that prevents it from being used.
WFS_BIO_DEVUSERERROR	The device is present but a person is preventing proper device operation. The application should suspend the device operation or remove the device from service until the Service Provider generates a device state change event indicating the condition of the device has changed e.g. the error is removed (WFS_BIO_DEVONLINE) or a permanent error condition has occurred (WFS_BIO_DEVHWERROR).
WFS_BIO_DEVBUSY	The device is busy and unable to process an Execute command at this time.
WFS_BIO_DEVFRAUDATTEMPT	The device is present but is inoperable because it has detected a fraud attempt.
WFS_BIO_DEVPOTENTIALFRAUD	The device has detected a potential fraud attempt and is capable of remaining in service. In this case the application should make the decision as to whether to take the device offline.

dwSubject
Specifies the state of the subject to be scanned (e.g. finger, palm, retina, etc~~)~~.) as one of the following values:

Value	Meaning
WFS_BIO_SUBJECTPRESENT	The subject to be scanned is on the scanning position.
WFS_BIO_SUBJECTNOTPRESENT	The subject to be scanned is not on the scanning position.
WFS_BIO_SUBJECTUNKNOWN	The subject to be scanned cannot be determined with the device in its current state (e.g. the value of *fwDevice* is WFS_BIO_DEVNODEVICE, WFS_BIO_DEVPOWEROFF, WFS_BIO_DEVOFFLINE, or WFS_BIO_DEVHWERROR).
WFS_BIO_SUBJECTNOTSUPPORTED	The physical device does not support the ability to report whether or not a subject is on the scanning position.

bCaptured
Indicates whether or not scanned biometric data has been captured using the WFS_CMD_BIO_READ command and is currently stored and ready for comparison. TRUE if data has been captured and is stored, FALSE if no scanned data is present. This will be set to FALSE when scanned data is cleared using the WFS_CMD_BIO_CLEAR command.

dwDataPersistence
Specifies the current data persistence mode. The data persistence mode controls how biometric data that has been captured using the WFS_CMD_BIO_READ command will be handled. For possible values see the description of the *fwPersistenceModes* capability field.

dwRemainingStorage
Specifies how much of the reserved storage specified by the *dwTemplateStorage* capability is remaining for the storage of templates in bytes. This will be zero if not reported.

lpszExtra
Pointer to a list of vendor-specific, or any other extended, information. The information is returned as a series of *"key=value"* strings so that it is easily extensible by Service Providers. Each string is null-terminated, with the final string terminating with two null characters. An empty list may be indicated by either a NULL pointer or a pointer to two consecutive null characters.

dwGuidLights [...]
Specifies the state of the guidance light indicators. The elements of this array can be accessed by using the predefined index values specified for the *dwGuidLights [ ]* field in the capabilities. Vendor specific guidance lights are defined starting from the end of the array. The maximum guidance light index is WFS_BIO_GUIDLIGHTS_MAX.

Specifies the state of the guidance light indicator as WFS_BIO_GUIDANCE_NOT_AVAILABLE, WFS_BIO_GUIDANCE_OFF or a combination of the following flags consisting of one type B, optionally one type C and optionally type D.

Value	Meaning	Type
WFS_BIO_GUIDANCE_NOT_AVAILABLE	The status is not available.	A
WFS_BIO_GUIDANCE_OFF	The light is turned off.	A
WFS_BIO_GUIDANCE_SLOW_FLASH	The light is blinking slowly.	B
WFS_BIO_GUIDANCE_MEDIUM_FLASH	The light is blinking medium frequency.	B
WFS_BIO_GUIDANCE_QUICK_FLASH	The light is blinking quickly.	B
WFS_BIO_GUIDANCE_CONTINUOUS	The light is turned on continuous (steady).	B
WFS_BIO_GUIDANCE_RED	The light is red.	C
WFS_BIO_GUIDANCE_GREEN	The light is green.	C
WFS_BIO_GUIDANCE_YELLOW	The light is yellow.	C
WFS_BIO_GUIDANCE_BLUE	The light is blue.	C
WFS_BIO_GUIDANCE_CYAN	The light is cyan.	C

WFS_BIO_GUIDANCE_MAGENTA	The light is magenta.	C
WFS_BIO_GUIDANCE_WHITE	The light is white.	C
WFS_BIO_GUIDANCE_ENTRY	The light is in the entry state.	D
WFS_BIO_GUIDANCE_EXIT	The light is in the exit state.	D

dwGuidLights [WFS_BIO_GUIDANCE_BIO]
Specifies the state of the guidance light indicator on the biometric device.

wDevicePosition
Specifies the device position. The device position value is independent of the *fwDevice* value, e.g. when the device position is reported as WFS_BIO_DEVICENOTINPOSITION, *fwDevice* can have any of the values defined above (including WFS_BIO_DEVONLINE or WFS_BIO_DEVOFFLINE). This value is one of the following values:

Value	Meaning
WFS_BIO_DEVICEINPOSITION	The device is in its normal operating position, or is fixed in place and cannot be moved.
WFS_BIO_DEVICENOTINPOSITION	The device has been removed from its normal operating position.
WFS_BIO_DEVICEPOSUNKNOWN	Due to a hardware error or other condition, the position of the device cannot be determined.
WFS_BIO_DEVICEPOSNOTSUPP	The physical device does not have the capability of detecting the position.

usPowerSaveRecoveryTime
Specifies the actual number of seconds required by the device to resume its normal operational state from the current power saving mode. This value is zero if either the power saving mode has not been activated or no power save control is supported.

wAntiFraudModule
Specifies the state of the anti-fraud module as one of the following values:

Value	Meaning
WFS_BIO_AFMNOTSUPP	No anti-fraud module is available.
WFS_BIO_AFMOK	Anti-fraud module is in a good state and no foreign device is detected.
WFS_BIO_AFMINOP	Anti-fraud module is inoperable.
WFS_BIO_AFMDEVICEDETECTED	Anti-fraud module detected the presence of a foreign device.
WFS_BIO_AFMUNKNOWN	The state of the anti-fraud module cannot be determined.

Error Codes Only the generic error codes defined in [Ref. 1] can be generated by this command.

Comments Applications which rely on the *lpszExtra* field may not be device or vendor-independent.

In the case where communication with the device has been lost, the *fwDevice* field will report WFS_BIO_DEVPOWEROFF when the device has been removed or WFS_BIO_DEVHWERROR if the communications are unexpectedly lost. All other fields should contain a value based on the following rules and priority:

1. Report the value as unknown.
2. Report the value as a general h/w error.
3. Report the value as the last known value.

4.2 WFS_INF_BIO_CAPABILITIES

Description This command retrieves the capabilities of the biometric device. It may also return vendor specific capability information.

Input Param None.

Output Param LPWFSBIOCAPS lpCaps;

```
typedef struct _wfs_bio_caps
     {
     WORD                    wClass;
     DWORD                   fwType;
     BOOL                    bCompound;
     USHORT                  usMaxCapture;
     DWORD                   dwTemplateStorage;
     DWORD                   fwDataFormats;
     DWORD                   fwEncryptionAlgorithms;
     WORD                    fwStorage;
     DWORD                   fwPersistenceModes;
     DWORD                   dwMatchSupported;
     WORD                    fwScanModes;
     WORD                    fwCompareModes;
     DWORD                   fwClearData;
     LPSTR                   lpszExtra;
     DWORD                   dwGuidLights[WFS_BIO_GUIDLIGHTS_SIZE];
     BOOL                    bPowerSaveControl;
     BOOL                    bAntiFraudModule;
     LPDWORD                 lpdwSynchronizableCommands;
     } WFSBIOCAPS, *LPWFSBIOCAPS;
```

wClass
Specifies the logical service class as WFS_SERVICE_CLASS_BIO.

fwType
Specifies the type of biometric device as a combination of the following flags:

Value	Meaning
WFS_BIO_TYPE_FACIAL_FEATURES	The biometric device supports facial recognition scanning.
WFS_BIO_TYPE_VOICE	The biometric device supports voice recognition.
WFS_BIO_TYPE_FINGERPRINT	The biometric device supports fingerprint scanning.
WFS_BIO_TYPE_FINGERVEIN	The biometric device supports finger vein scanning.
WFS_BIO_TYPE_IRIS	The biometric device supports iris scanning.
WFS_BIO_TYPE_RETINA	The biometric device supports retina scanning.
WFS_BIO_TYPE_HAND_GEOMETRY	The biometric device supports hand geometry scanning.
WFS_BIO_TYPE_THERMAL_FACE	The biometric device supports thermal face image scanning.
WFS_BIO_TYPE_THERMAL_HAND	The biometric device supports thermal hand image scanning.
WFS_BIO_TYPE_PALM_VEIN	The biometric device supports palm vein scanning.
WFS_BIO_TYPE_SIGNATURE	The biometric device supports signature scanning.

bCompound
Specifies whether the biometric device is part of a compound device.

usMaxCapture
Specifies the maximum number of times that the device can attempt to capture biometric data during a WFS_CMD_BIO_READ command. If this is zero then the device or service provider determines how many captures will be attempted.

dwTemplateStorage
Specifies the storage space that is reserved on the device for the storage of templates in bytes. This will be set to zero if not reported or unknown.

fwDataFormats
Specifies the supported biometric raw data and template data formats reported as a combination of the following flags:

Value	Meaning
WFS_BIO_ISOFID	Raw ISO FID format [Ref. 4].
WFS_BIO_ISOFMD	ISO FMD template format [Ref. 5].
WFS_BIO_ANSIFID	Raw ANSI FID format [Ref. 2].
WFS_BIO_ANSIFMD	ANSI FMD template format [Ref. 3].
WFS_BIO_QSO	Raw QSO image format.
WFS_BIO_WSQ	WSQ image format.
WFS_BIO_RESERVED_RAW_1	Reserved for a vendor-defined Raw format.
WFS_BIO_RESERVED_TEMPLATE_1	Reserved for a vendor-defined Template format.
WFS_BIO_RESERVED_RAW_2	Reserved for a vendor-defined Raw format.
WFS_BIO_RESERVED_TEMPLATE_2	Reserved for a vendor-defined Template format.
WFS_BIO_RESERVED_RAW_3	Reserved for a vendor-defined Raw format.
WFS_BIO_RESERVED_TEMPLATE_3	Reserved for a vendor-defined Template format.

fwEncryptionAlgorithms
Supported encryption algorithms will be reported as a combination of the following flags, or WFS_BIO_CRYPT_NONE if no encryption algorithms are supported:

Value	Meaning
WFS_BIO_CRYPT_TRIDESECB	Triple DES with Electronic Code Book.
WFS_BIO_CRYPT_TRIDESCBC	Triple DES with Cipher Block Chaining.
WFS_BIO_CRYPT_TRIDESCFB	Triple DES with Cipher Feed Back.
WFS_BIO_CRYPT_RSA	RSA Encryption.

fwStorage
Indicates whether or not biometric template data can be stored securely as a combination of the following flags, or WFS_BIO_STORAGE_NONE if Biometric template data is not stored in the device:

Value	Meaning
WFS_BIO_STORAGE_SECURE	Biometric template data is securely stored as encrypted data.
WFS_BIO_STORAGE_CLEAR	Biometric template data is stored unencrypted in the device.

fwPersistenceModes
Specifies which data persistence modes can be set using the WFS_CMD_BIO_SET_DATA_PERSISTENCE command. This applies specifically to the biometric data that has been captured using the WFS_CMD_BIO_READ command. A value of WFS_BIO_PS_NONE indicates that persistence is entirely under device control and cannot be set, otherwise, valid values are a combination of the following flags:

Value	Meaning
WFS_BIO_PS_PERSIST	Biometric data captured using the WFS_CMD_BIO_READ command can persist until all XFS sessions are closed, the device is power failed or rebooted, or the WFS_CMD_BIO_READ command is requested again. This captured biometric data can also be explicitly cleared using the WFS_CMD_BIO_CLEAR or WFS_CMD_BIO_RESET commands.

WFS_BIO_PS_AUTOCLEAR	Captured biometric data will not persist. Once the data has been either returned in the WFS_CMD_BIO_READ command or used by the WFS_CMD_BIO_MATCH command, then the data is cleared from the device.

dwMatchSupported
Specifies if matching is supported using the WFS_CMD_BIO_MATCH and/or WFS_CMD_BIO_SET_MATCH commands. This will be one of the following values:

Value	Meaning
WFS_BIO_MTC_NONE	The device does not support matching.
WFS_BIO_MTC_STORED_MATCH	The device scans biometric data using the WFS_CMD_BIO_READ command and stores it, then the scanned data can be compared with imported biometric data using the WFS_CMD_BIO_MATCH command (See section 7.2 - Biometric Match Command Flow – Separate Scan and Match).
WFS_BIO_MTC_COMBINED_MATCH	The device scans biometric data and performs a match against imported biometric data as a single operation. The WFS_CMD_BIO_SET_MATCH command must be called before the WFS_CMD_BIO_READ command in order to set the matching criteria. Then the WFS_CMD_BIO_MATCH command can be called to return the result (See section 7.3 - Biometric Match Command Flow – Combined Scan and Match).

fwScanModes
Specifies the modes that the WFS_CMD_BIO_READ command can be used for, as a combination of the following flags:

Value	Meaning
WFS_BIO_MODE_SCAN	The WFS_CMD_BIO_READ command can be used to scan data only, for example to enroll a user or collect data for matching in an external biometric system.
WFS_BIO_MODE_MATCH	The WFS_CMD_BIO_READ command can be used to scan data for a match operation using the WFS_CMD_BIO_MATCH command.

fwCompareModes
Specifies the type of match operations that can be performed as a combination of the following flags. A value of WFS_BIO_COMP_NONE indicates that matching is not supported:

Value	Meaning
WFS_BIO_COMP_VERIFY	The biometric data can be compared as a one to one verification operation.
WFS_BIO_COMP_IDENTIFY	The biometric data can be compared as a one to many identification operation.

fwClearData
Specifies the type of data that can be cleared from storage using the WFS_CMD_BIO_CLEAR or WFS_CMD_BIO_RESET commands as either WFS_BIO_CLR_NONE or a combination of the following flags:

Value	Meaning
WFS_BIO_CLR_SCANNEDDATA	Raw image data that has been scanned using the WFS_CMD_BIO_READ command can be cleared.
WFS_BIO_CLR_IMPORTEDDATA	Template data that was imported using the WFS_CMD_BIO_IMPORT command can be cleared.
WFS_BIO_CLR_SETMATCHDATA	Match criteria data that was set using the WFS_CMD_SET_MATCH command can be cleared.

lpszExtra
Pointer to a list of vendor-specific, or any other extended, information. The information is returned as a series of *"key=value"* strings so that it is easily extensible by Service Providers. Each string is null-terminated, with the final string terminating with two null characters. An empty list may be indicated by either a NULL pointer or a pointer to two consecutive null characters.

dwGuidLights [...]
Specifies which guidance lights are available. A number of guidance light positions are defined below. Vendor specific guidance lights are defined starting from the end of the array. The maximum guidance light index is WFS_BIO_GUIDLIGHTS_MAX.

In addition to supporting specific flash rates and colors, some guidance lights also have the capability to show directional movement representing "exit".

The elements of this array are specified as a combination of the following flags and indicate all of the possible flash rates (type B) colors (type C) and directions (type D) that the guidance light indicator is capable of handling. If the guidance light indicator only supports one color, then no value of type C is returned. If the guidance light indicator does not support direction, then no value of type D is returned. A value of WFS_BIO_GUIDANCE_NOT_AVAILABLE indicates that the device has no guidance light indicator or the device controls the light directly with no application control possible.

Value	Meaning	Type
WFS_BIO_GUIDANCE_NOT_AVAILABLE	There is no guidance light control available at this position.	A
WFS_BIO_GUIDANCE_OFF	The light can be off.	B
WFS_BIO_GUIDANCE_SLOW_FLASH	The light can blink slowly.	B
WFS_BIO_GUIDANCE_MEDIUM_FLASH	The light can blink medium frequency.	B
WFS_BIO_GUIDANCE_QUICK_FLASH	The light can blink quickly.	B
WFS_BIO_GUIDANCE_CONTINUOUS	The light can be continuous (steady).	B
WFS_BIO_GUIDANCE_RED	The light can be red.	C
WFS_BIO_GUIDANCE_GREEN	The light can be green.	C
WFS_BIO_GUIDANCE_YELLOW	The light can be yellow.	C
WFS_BIO_GUIDANCE_BLUE	The light can be blue.	C
WFS_BIO_GUIDANCE_CYAN	The light can be cyan.	C
WFS_BIO_GUIDANCE_MAGENTA	The light can be magenta.	C
WFS_BIO_GUIDANCE_WHITE	The light can be white.	C
WFS_BIO_GUIDANCE_ENTRY	The light can be in the entry state.	D
WFS_BIO_GUIDANCE_EXIT	The light can be in the exit state.	D

dwGuidLights [WFS_BIO_GUIDANCE_BIO]
Specifies whether the guidance light indicator on the biometric device is available.

bPowerSaveControl
Specifies whether power saving control is available. This can either be TRUE if available or FALSE if not available.

bAntiFraudModule
Specifies whether the anti-fraud module is available. This can either be TRUE if available or FALSE if not available.

	lpdwSynchronizableCommands Pointer to a zero-terminated list of DWORDs which contains the execute command IDs that can be synchronized. If no execute command can be synchronized, then this parameter will be NULL.
Error Codes	Only the generic error codes defined in [Ref. 1] can be generated by this command.
Comments	Applications which rely on the *lpszExtra* field may not be device or vendor-independent.

4.3 WFS_INF_BIO_STORAGE_INFO

Description This command is used to obtain information regarding the number and format of biometric templates that have been imported using the WFS_CMD_BIO_IMPORT command.

Input Param None.

Output Param LPWFSBIOSTORAGELIST lpStorageList;

```
typedef struct _wfs_bio_storage_list
     {
     USHORT                usCount;
     LPWFSBIOSTORAGE       *lppStorageList;
     } WFSBIOSTORAGELIST, *LPWFSBIOSTORAGELIST;
```

usCount
Specifies the number of WFSBIOSTORAGE structures returned in *lppStorageList*.

lppStorageList
Pointer to an array of pointers to WFSBIOSTORAGE structures:

```
typedef struct _wfs_bio_storage
     {
     USHORT                     usIdentifier;
     LPWFSBIODATATYPE           lpType;
     } WFSBIOSTORAGE, *LPWFSBIOSTORAGE;
```

usIdentifier
A unique number which identifies the template.

lpType
Pointer to a WFSBIODATATYPE structure that specifies the biometric data type of the template data.

```
typedef struct _wfs_bio_data_type
     {
     DWORD                dwFormat;
     DWORD                dwAlgorithm;
     LPSTR                lpszKeyName;
     } WFSBIODATATYPE, *LPWFSBIODATATYPE;
```

dwFormat
Specifies the format of the template data. For possible values see the description of the *fwDataFormats* capability field.

dwAlgorithm
Specifies the encryption algorithm. For possible values see the description of the *fwEncryptionAlgorithms* capability field. This value is WFS_BIO_CRYPTNONE if *lpszKeyName* is NULL.

lpszKeyName
Specifies the name of the key that is used to encrypt the biometric data. This value is NULL if the biometric data is not encrypted.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_BIO_NOIMPORTEDDATA	No data to return. Typically means that no data has been imported using the WFS_CMD_BIO_IMPORT command.

Comments None.

4.4 WFS_INF_BIO_KEY_INFO

Description This command returns detailed information about the keys in the biometric module, including symmetric and asymmetric keys, that can be used for biometric data encryption and decryption. This command will also return information on all keys loaded during manufacture that can be used by applications for biometric data encryption and decryption.

Input Param LPSTR lpszKeyName;

lpszKeyName
Specifies a string which identifies the name of the key for which detailed information is requested. This string value is terminated with a null character. If *lpszKeyName* is set to NULL, detailed information about all the keys in the biometric module that can be used for biometric data encryption or decryption are returned.

Output Param LPWFSBIOKEYINFO *lppKeyInfo;

Pointer to a null-terminated array of pointers to WFSBIOKEYINFO structures.

```
typedef struct _wfs_bio_key_info
     {
     LPSTR                    lpszKeyName;
     DWORD                    dwUse;
     BOOL                     bLoaded;
     } WFSBIOKEYINFO, *LPWFSBIOKEYINFO;
```

lpszKeyName
Specifies the name of the key.

dwUse
Specifies the type of access for which the key is used as a combination of the following flags:

Value	Meaning
WFS_BIO_USECRYPT	Key can be used for symmetric encryption/decryption.
WFS_BIO_USERSAPUBLIC	Key is used as a public key for RSA asymmetric encryption.

bLoaded
Specifies whether the key has been loaded.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_BIO_KEYNOTFOUND	The specified key name is not found.

Comments When the biometric module contains a public/private key-pair, only the public part of the key will be reported. In order to obtain the public key data, it is recommended to use the XFS PIN device class WFS_CMD_PIN_GET_CERTIFICATE or WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM command.

For biometric modules that can support application key loading, it is recommended to use the XFS PIN device class for key management functionality.

5. Execute Commands

5.1 WFS_CMD_BIO_READ

Description

This command enables the device for biometric scanning, then captures and optionally returns biometric data. A WFS_EXEE_BIO_PRESENTSUBJECT event will be sent to notify the application when it is ready to begin scanning and a WFS_EXEE_BIO_SUBJECTDETECTED event sent for each scanning attempt. The *usNumCaptures* input parameter specifies how many captures should be attempted, unless it is zero in which case the device itself will determine this. Once this command has successfully captured biometric raw data it will complete with WFS_SUCCESS.

The WFS_CMD_BIO_READ command has two purposes:

Scanning: The biometric data that is captured into the device can be processed into biometric template data and returned as an output parameter for enrollment or storage elsewhere, e.g. on a server or smart card.

Matching: The biometric data that is captured into the device can be used for subsequent matching. Once data has been scanned into the device it can be compared to existing biometric templates that have been imported using the WFS_CMD_BIO_IMPORT command in order to allow verification or identification of an individual. The *dwMatchSupported* capability indicates if the WFS_CMD_BIO_MATCH command can be used for matching, otherwise the matching must be done externally, e.g. on a server or smart card.

In either case the data that has been scanned into the device will be persistent according to the current persistence mode as reported by the *dwDataPersistence* status field.

Examples of the above use cases are detailed in the appendix in section 7. Biometric Device Command Flows – Application Guidelines.

Input Param

LPWFSBIOREAD lpRead;

```
typedef struct _wfs_bio_read
     {
     USHORT              usCount;
     LPWFSBIODATATYPE    *lppTypes;
     USHORT              usNumCaptures;
     USHORT              usMode;
     } WFSBIOREAD, *LPWFSBIOREAD;
```

usCount
Specifies the number of LPWFSBIODATATYPE structures returned in *lppTypes*.

lppTypes
Pointer to an array of pointers to WFSBIODATATYPE structures, each element of which represents the data type(s) in which the data should be returned in the *lpReadData* output parameter. If no data is to be returned *lppTypes* should be set to NULL. Single or multiple formats can be returned, or no data can be returned in the case where the scan is to be followed by a subsequent matching operation. For a description of the WFSBIODATATYPE type refer to the description in the WFS_INF_BIO_STORAGE_INFO command.

usNumCaptures
This field indicates the number of times to attempt capture of the biometric data from the subject. If this is zero, then the device determines how many attempts will be made. The maximum number of captures possible is indicated by the *usMaxCapture* capability.

usMode
This optional field indicates the reason why the WFS_CMD_BIO_READ command has been issued, in order to allow for any necessary optimization. Possible values are detailed in the *fwScanModes* capability.

Output Param LPWFSBIOREADDATA lpReadData;

If the LPWFSBIOREAD.*lppTypes* input parameter is NULL then no data will be returned and the *lpReadData* output parameter will be NULL. Otherwise the *lpReadData* output parameter will be as follows:

```
typedef struct _wfs_bio_read_data
     {
     USHORT               usCount;
     LPWFSBIODATA         *lppBioDataList;
     } WFSBIOREADDATA, *LPWFSBIOREADDATA;
```

usCount
Specifies the number of LPWFSBIODATA structures returned in *lppBioDataList.*

lppBioDataList
Pointer to an array of pointers to WFSBIODATA structures. The data type LPWFSBIODATA is used to contain the returned data and its format. It is defined as follows:

```
typedef struct _wfs_bio_data
     {
     LPWFSBIODATATYPE     lpType;
     LPWFSXBIODATA        lpxData;
     } WFSBIODATA, *LPWFSBIODATA;
```

lpType
This field is used to indicate the biometric data type of the template data contained in *lpxData*. For a description of the WFSBIODATATYPE type refer to the description in the WFS_INF_BIO_STORAGE_INFO command.

lpxData
Pointer to a WFSXBIODATA data type containing the binary data stream.

```
typedef struct _wfs_bio_hex_data
     {
     USHORT               usLength;
     LPBYTE               lpbData;
     } WFSXBIODATA, *LPWFSXBIODATA;
```

usLength
Length of the byte stream pointed to by *lpbData.*

lpbData
Pointer to the binary data stream.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_BIO_READFAILED	Module was unable to complete the scan operation.
WFS_ERR_BIO_MODENOTSUPP	The input parameter *usMode* contains a value that is not supported.
WFS_ERR_BIO_FORMATNOTSUPP	The format specified is valid but not supported. A list of the supported values can be obtained through the *fwDataFormats* capability field.
WFS_ERR_BIO_KEYNOTFOUND	The specified key name is not found.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated as a result of this command:

Value	Meaning
WFS_EXEE_BIO_PRESENTSUBJECT	This event notifies the application when the device is ready for the user to present the subject to be captured to the biometric scanner, and can be sent as many times as specified by *usNumCaptures* or as many times as the device supports.
WFS_EXEE_BIO_SUBJECTDETECTED	The device has detected a subject and an attempt to capture biometric data has been performed.
WFS_EXEE_BIO_REMOVESUBJECT	This event notifies an application when the subject should be removed from the device for the next scan attempt.
WFS_SRVE_BIO_SUBJECTREMOVED	The device has detected that the subject has been removed from the biometric sensor.
WFS_SRVE_BIO_DATACLEARED	This event notifies an application that the data which has been captured and returned has been automatically cleared from the device (status *wDataPersistence* == WFS_BIO_PS_AUTOCLEAR).
WFS_EXEE_BIO_ORIENTATION	This event notifies an application that the user has presented the subject to the biometric sensor in an incorrect orientation. The application should prompt the user to correct it.

Comments None.

5.2 WFS_CMD_BIO_IMPORT

Description This command imports a list of biometric template data structures into the device for later comparison with biometric data scanned using the WFS_CMD_BIO_READ command. Normally this data is read from the chip on a customer's card or provided by the host system. Data that has been imported is available until a WFS_CMD_BIO_CLEAR command is called. If template data has been previously imported using a call to WFS_CMD_BIO_IMPORT, then it is overwritten. This data is not persistent across power fails.

Input Param LPWFSBIOIMPORTDATA lpImportData;

```
typedef struct _wfs_bio_import_data
     {
     USHORT                usCount;
     LPWFSBIODATA          *lppBioDataList;
     } WFSBIOIMPORTDATA, *LPWFSBIOIMPORTDATA;
```

usCount
Specifies the number of LPWFSBIODATA structures in *lppBioDataList*. Note that if a simple one-to-one verification comparison is to be performed using the WFS_CMD_BIO_MATCH command then *usCount* should be 1 and *lppBioDataList* will point to an array of only one WFSBIODATA structure.

lppBioDataList
Pointer to an array of pointers to the WFSBIODATA structures to be imported. For a description of the WFSBIODATA type refer to the description in the WFS_CMD_BIO_READ command.

Output Param LPWFSBIOSTORAGELIST lpStorageList;

A list of the biometric template data structures that were successfully imported. For the structure definition of the WFSBIOSTORAGELIST see the WFS_INF_BIO_STORAGE_INFO command.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_BIO_INVALIDDATA	The data that was imported was malformed or invalid. No data has been imported into the device. The presence of any previously loaded templates can be checked for using the WFS_INF_BIO_STORAGE_INFO command.
WFS_ERR_BIO_FORMATNOTSUPP	The format of the biometric data that was specified is not supported. No data has been imported into the device. A list of the supported values can be obtained through the *fwDataFormats* capability field.
WFS_ERR_BIO_CAPACITYEXCEEDED	An attempt has been made to import more templates than the maximum reserved storage space available. The maximum storage space available is reported in the capability *dwTemplateStorage*. No data has been imported into the device. The amount of storage remaining is reported in the WFSBIOSTATUS.*dwRemainingStorage* status field.
WFS_ERR_BIO_KEYNOTFOUND	The specified key name is not found.

Events None.

Comments None.

5.3 WFS_CMD_BIO_MATCH

Description This command returns the result of a comparison between data that has been scanned using the WFS_CMD_BIO_READ command and template data that has been imported using the WFS_CMD_BIO_IMPORT command. The comparison may be performed by this command or the WFS_CMD_BIO_READ, this command is responsible for returning the result. WFS_SUCCESS is returned if the device has been able to successfully compare the data, however this does not necessarily mean that the data matched.

If the capability *dwMatchSupported* value == WFS_BIO_MTC_COMBINED_MATCH then the device performs a combined scan and match operation, and the WFS_CMD_BIO_SET_MATCH must be called before this command in order to set the matching criteria. In this case if WFS_CMD_BIO_SET_MATCH has not been called then this command will fail with WFS_ERR_SEQUENCE_ERROR.

If the capability *dwMatchSupported* == WFS_BIO_STORED_MATCH then the device will scan data using the WFS_CMD_BIO_READ command and store it, then the data can be compared with imported biometric data using the WFS_CMD_BIO_MATCH command.

This command can be used in two modes of operation: Verification or Identification, as indicated by the *usCompareMode* input parameter. The two modes of operation are described below:

Verification (*usCompareMode* == WFS_BIO_VERIFY) :
In this case a one to one comparison is performed and the *usMaximum* input parameter is ignored. The data that has been scanned previously using the WFS_CMD_BIO_READ command is compared with a single template that has been imported using the WFS_CMD_BIO_IMPORT command. If there is a successful match then the *usConfidenceLevel* output parameter can be used to determine the quality of the match and will be in the range 0 – 100, where 100 represents an exact match and 0 represents no match.

Identification (*usCompareMode* == WFS_BIO_IDENTIFY) :
In this case a one to many comparison is performed. The data that has been scanned previously using the WFS_CMD_BIO_READ command is compared with multiple templates that have been imported using the WFS_CMD_BIO_IMPORT command. The input parameter *usMaximum* is used to specify the maximum number of matches to return: a smaller number can make execution faster. The required degree of matching similarity can be controlled using the *usThreshold* parameter which is used to control the frequency of false positive and false negative matching errors. The value of *usThreshold* represents the criteria as to what constitutes a successful match and is in the range 0 – 100, where 100 represents an exact match and 0 represents no match. If for example, *usThreshold* is set to 75 then only results with a matching score equal to or greater than 75 are returned. The matching candidate list is returned in the *lpMatchResult* output parameter sorted in order of highest score. The higher the value of *usConfidenceLevel* the closer the candidate is to the beginning of the list, with the best match being the first candidate in the list. Note that where the number of templates that match the criteria of the threshold are greater than *usMaximum*, only the *usMaximum* templates with the highest score will be returned.

Input Param LPWFSBIOMATCH lpMatch;

```
typedef struct _wfs_bio_match
     {
     USHORT          usCompareMode;
     USHORT          usIdentifier;
     USHORT          usMaximum;
     USHORT          usThreshold;
     } WFSBIOMATCH, *LPWFSWFSBIOMATCH;
```

usCompareMode
Specifies the type of match operation that is being done. Valid values are:

Value	Meaning
WFS_BIO_COMP_VERIFY	The biometric data will be compared as a one to one verification operation.
WFS_BIO_COMP_IDENTIFY	The biometric data will be compared as a one to many identification operation.

usIdentifier
In the case where *usCompareMode* is WFS_BIO_COMP_VERIFY this parameter corresponds to a template that has been imported by a previous call to the WFS_CMD_BIO_IMPORT command. If *usCompareMode* is WFS_BIO_COMP_IDENTIFY a comparison is performed against all of the imported templates, in which case this parameter is ignored.

usMaximum
Specifies the maximum number of matches to return. In the case where *usCompareMode* is WFS_BIO_COMP_VERIFY this parameter is ignored.

usThreshold
Specifies the minimum matching confidence level necessary for the candidate to be included in the results. This value should be in the range of 0 to 100, where 100 represents an exact match and 0 represents no match.

Output Param LPWFSBIOMATCHRESULT lpMatchResult;

```
typedef struct _wfs_bio_match_result
     {
     USHORT                    usCount;
     LPWFSBIOCANDIDATE         *lppTemplateList;
     } WFSBIOMATCHRESULT, *LPWFSBIOMATCHRESULT;
```

usCount
Specifies the number of LPWFSBIOCANDIDATE structures returned in *lppTemplateList*. This will always be 1 where a verification is being performed and a successful match has been made.

lppTemplateList
Pointer to an array of pointers to LPWFSBIOCANDIDATE structures. This will be an empty list and *usCount* will be zero if the WFS_CMD_BIO_MATCH operation completes with no match found. If there are matches found, *lppTemplateList* contains all of the matching templates in order of confidence level, with the highest score first. Note that where the number of templates that match the input criteria of the threshold are greater than *usMaximum*, only the *usMaximum* templates with the highest scores will be returned.

```
typedef struct _wfs_bio_candidate
     {
     USHORT               usConfidenceLevel;
     USHORT               usIdentifier;
     LPWFSBIODATA         lpData;
     } WFSBIOCANDIDATE, * LPWFSBIOCANDIDATE;
```

usConfidenceLevel
Specifies the level of confidence for the match found. This value is in a scale of 0 - 100, where 0 is no match and 100 is an exact match. The minimum value will be that which was set by the *usThreshold* input parameter.

usIdentifier
A unique number that positively identifies the biometric template data. This corresponds to the list of template identifiers returned by the WFS_INF_BIO_STORAGE_INFO command.

lpData
Contains the biometric template data that was matched. This data may be used as justification for the biometric data match or confidence level. This pointer is NULL if no additional comparison data is returned. For a description of the WFSBIODATA type refer to the description in the WFS_CMD_BIO_READ command.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_BIO_NOIMPORTEDDATA	The command failed because no data was imported previously using the WFS_CMD_BIO_IMPORT_DATA command.
WFS_ERR_BIO_INVALIDIDENTIFIER	The command failed because data was imported but *usIdentifier* was not found.
WFS_ERR_BIO_MODENOTSUPP	The type of match specified in *usCompareMode* is not supported.
WFS_ERR_BIO_NOCAPTUREDDATA	No captured data is present. Typically means that the WFS_CMD_BIO_READ command has not been called, or the captured data has been cleared using the WFS_CMD_BIO_CLEAR command.
WFS_ERR_BIO_INVALIDCOMPAREMODE	The compare mode specified by the *usCompareMode* input parameter is not supported.
WFS_ERR_BIO_INVALIDTHRESHOLD	The *usThreshold* input parameter is greater than the maximum allowed of 100.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated as a result of this command:

Value	Meaning
WFS_SRVE_BIO_DATACLEARED	This event notifies an application that the data which has been captured and returned has been automatically cleared from the device (status *dwDataPersistence* == WFS_BIO_PS_AUTOCLEAR).

Comments None.

5.4 WFS_CMD_BIO_SET_MATCH

Description This command is used for devices which need to know the match criteria data for the WFS_CMD_BIO_MATCH command before any biometric scanning is performed by the WFS_CMD_BIO_READ command. WFS_CMD_BIO_READ and WFS_CMD_BIO_MATCH should be called after this command. For all other devices WFS_ERR_UNSUPP_COMMAND will be returned here.

If the capability *dwMatchSupported* == WFS_BIO_MTC_COMBINED_MATCH then this command is mandatory. If it is not called first, the WFS_CMD_BIO_MATCH command will fail with the generic error WFS_ERR_SEQUENCE_ERROR. The data set using this command is not persistent across power failures.

Input Param LPWFSBIOMATCH lpMatch;

See WFS_CMD_BIO_MATCH for details.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_BIO_INVALIDIDENTIFIER	The command failed because data was imported but *usIdentifier* was not found.
WFS_ERR_BIO_MODENOTSUPP	The type of match specified in *usCompareMode* is not supported.
WFS_ERR_BIO_NOIMPORTEDDATA	The command failed because no data was imported previously using the WFS_CMD_BIO_IMPORT_DATA command.
WFS_ERR_BIO_INVALIDTHRESHOLD	The *usThreshold* input parameter is greater than the maximum allowed of 100.

Events None.

Comments None.

5.5 WFS_CMD_BIO_CLEAR

Description This command can be used to clear stored data. In the case where there is no stored data to clear this command completes with WFS_SUCCESS.

Input Param LPWFSBIOCLEAR lpClear;

```
typedef struct _wfs_bio_clear
     {
     DWORD      fwClearData;
     } WFSBIOCLEAR, *LPWFSBIOCLEAR;
```

fwClearData
This parameter indicates the type of data to be cleared from storage as a combination of flags. If this is set to zero then all stored data will be cleared. For a list of possible values see the *fwClearData* capability.

Output Param None.

Error Codes Only the generic error codes defined in [Ref. 1] can be generated by this command.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated as a result of this command:

Value	Meaning
WFS_SRVE_BIO_DATACLEARED	This event notifies an application that data has been cleared from the device.

Comments None.

5.6 WFS_CMD_BIO_RESET

Description This command is used by the application to perform a hardware reset which will attempt to return the biometric device to a known good state.

Input Param LPWFSBIORESET lpResetIn;

```
typedef struct _wfs_bio_reset
     {
     DWORD      fwClearData;
     } WFSBIORESET, *LPWFSBIORESET;
```

fwClearData
This parameter indicates the type of data to be cleared from storage as a combination of flags. If this is set to zero then all stored data will be cleared. For a list of possible values see the *fwClearData* capability.

Output Param None.

Error Codes Only the generic error codes defined in [Ref. 1] can be generated by this command.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated as a result of this command:

Value	Meaning
WFS_SRVE_BIO_DATACLEARED	This event notifies an application that data has been cleared from the device.

Comments This command is used by an application control program to cause a device to reset itself to a known good condition.

5.7 WFS_CMD_BIO_SET_DATA_PERSISTENCE

Description This command is used to set the persistence mode. This controls how the biometric data is persisted after a WFS_CMD_BIO_READ command. The data can be persisted for use by subsequent commands, or it can be automatically cleared.

Input Param LPWFSBIOPERSISTDATA lpPersistDataIn;

```
typedef struct _wfs_bio_persist_data
     {
     DWORD      dwPersistenceMode;
     } WFSBIOPERSISTDATA, *LPWFSBIOPERSISTDATA;
```

dwPersistenceMode
Specifies the data persistence mode. This controls how biometric data that has been captured using the WFS_CMD_BIO_READ command will persist. Available modes are reported in the *fwPersistenceModes* capability field. This value itself is persistent.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_BIO_MODENOTSUPP	The command failed because a mode was specified which is not supported.

Events Only the generic events defined in [Ref. 1] can be generated by this command.

Comments When using this command to maintain data persistence, applications should ensure that a customer’s biometric data is cleared after they have completed all their transactions. The data can be explicitly cleared using the WFS_CMD_BIO_CLEAR command.

5.8 WFS_CMD_BIO_SET_GUIDANCE_LIGHT

Description This command is used to set the status of the BIO guidance lights. This includes defining the flash rate, the color and the direction. When an application tries to use a color or direction that is not supported then the Service Provider will return the generic error WFS_ERR_UNSUPP_DATA.

Input Param LPWFSBIOSETGUIDLIGHT lpSetGuidLight;

```
typedef struct _wfs_bio_set_guidlight
     {
     WORD            wGuidLight;
     DWORD           dwCommand;
     } WFSBIOSETGUIDLIGHT, *LPWFSBIOSETGUIDLIGHT;
```

wGuidLight
Specifies the index of the guidance light to set as one of the values defined within the capabilities section.

dwCommand
Specifies the state of the guidance light indicator as WFS_BIO_GUIDANCE_OFF or a combination of the following flags consisting of one type B, optionally one type C and optionally one type D. If no value of type C is specified, then the default color is used. The Service Provider determines which color is used as the default color.

Value	Meaning	Type
WFS_BIO_GUIDANCE_OFF	The light indicator is turned off.	A
WFS_BIO_GUIDANCE_SLOW_FLASH	The light indicator is set to flash slowly.	B
WFS_BIO_GUIDANCE_MEDIUM_FLASH	The light indicator is set to flash medium frequency.	B
WFS_BIO_GUIDANCE_QUICK_FLASH	The light indicator is set to flash quickly.	B
WFS_BIO_GUIDANCE_CONTINUOUS	The light indicator is turned on continuously (steady).	B
WFS_BIO_GUIDANCE_RED	The light indicator color is set to red.	C
WFS_BIO_GUIDANCE_GREEN	The light indicator color is set to green.	C
WFS_BIO_GUIDANCE_YELLOW	The light indicator color is set to yellow.	C
WFS_BIO_GUIDANCE_BLUE	The light indicator color is set to blue.	C
WFS_BIO_GUIDANCE_CYAN	The light indicator color is set to cyan.	C
WFS_BIO_GUIDANCE_MAGENTA	The light indicator color is set to magenta.	C
WFS_BIO_GUIDANCE_WHITE	The light indicator color is set to white.	C
WFS_BIO_GUIDANCE_ENTRY	The light indicator is set to the entry state.	D
WFS_BIO_GUIDANCE_EXIT	The light indicator is set to the exit state.	D

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_BIO_INVALID_PORT	An attempt to set a guidance light to a new value was invalid because the guidance light does not exist.

Events Only the generic events defined in [Ref. 1] can be generated by this command:

Comments The slow and medium flash rates must not be greater than 2.0 Hz. It should be noted that in order to comply with American Disabilities Act guidelines only a slow or medium flash rate must be used.

5.9 WFS_CMD_BIO_POWER_SAVE_CONTROL

Description	This command activates or deactivates the power saving mode.

If the Service Provider receives another execute command while in power saving mode, the Service Provider automatically exits the power saving mode, and executes the requested command. If the Service Provider receives an information command while in power saving mode, the Service Provider will not exit the power saving mode.

Input Param LPWFSBIOPOWERSAVECONTROL lpPowerSaveControl;

```
typedef struct _wfs_bio_power_save_control
     {
     USHORT          usMaxPowerSaveRecoveryTime;
     } WFSBIOPOWERSAVECONTROL, *LPWFSBIOPOWERSAVECONTROL;
```

usMaxPowerSaveRecoveryTime
Specifies the maximum number of seconds in which the device must be able to return to its normal operating state when exiting power save mode. The device will be set to the highest possible power save mode within this constraint. If *usMaxPowerSaveRecoveryTime* is set to zero, then the device will exit the power saving mode.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_BIO_POWERSAVETOOSHORT	The power saving mode has not been activated because the device is not able to resume from the power saving mode within the specified *usMaxPowerSaveRecoveryTime* value.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_BIO_POWER_SAVE_CHANGE	The power save recovery time has changed.

Comments None.

5.10 WFS_CMD_BIO_SYNCHRONIZE_COMMAND

Description This command is used to reduce response time of a command (e.g. for synchronization with display) as well as to synchronize actions of the different device classes. This command is intended to be used only on hardware which is capable of synchronizing functionality within a single device class or with other device classes.

The list of execute commands which this command supports for synchronization is retrieved in the *lpdwSynchronizableCommands* parameter of the WFS_INF_BIO_CAPABILITIES.

This command is optional, i.e. any other command can be called without having to call it in advance. Any preparation that occurs by calling this command will not affect any other subsequent command. However, any subsequent execute command other than the one that was specified in the *dwCommand* input parameter will execute normally and may invalidate the pending synchronization. In this case the application should call the WFS_CMD_BIO_SYNCHRONIZE_COMMAND again in order to start a synchronization.

Input Param LPWFSBIOSYNCHRONIZECOMMAND lpSynchronizeCommand;

```
typedef struct _wfs_bio_synchronize_command
     {
     DWORD           dwCommand;
     LPVOID          lpCmdData;
     } WFSBIOSYNCHRONIZECOMMAND, *LPWFSBIOSYNCHRONIZECOMMAND;
```

dwCommand
The command ID of the command to be synchronized and executed next.

lpCmdData
Pointer to data or a data structure that represents the parameter that is normally associated with the command that is specified in *dwCommand*. For example, if *dwCommand* is WFS_CMD_BIO_READ then *lpCmdData* will point to a WFSBIOREAD structure. This parameter can be NULL if no command input parameter is needed or if this detail is not needed to synchronize for the command.

It will be device-dependent whether the synchronization is effective or not in the case where the application synchronizes for a command with this command specifying a parameter but subsequently executes the synchronized command with a different parameter. This case should not result in an error; however, the preparation effect could be different from what the application expects. The application should, therefore, make sure to use the same parameter between *lpCmdData* of this command and the subsequent corresponding execute command.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_BIO_COMMANDUNSUPP	The command specified in the *dwCommand* field is not supported by the Service Provider.
WFS_ERR_BIO_SYNCHRONIZEUNSUPP	The preparation for the command specified in the *dwCommand* with the parameter specified in the *lpCmdData* is not supported by the Service Provider.

Events Only the generic events defined in [Ref. 1] can be generated by this command.

Comments For sample flows of this synchronization see the [Ref 1] Appendix C.

6. Events

6.1 WFS_EXEE_BIO_PRESENTSUBJECT

Description	This execute event is generated to notify the application when the device is ready for a user to present the subject to be captured to the biometric scanner, for example, placing a finger on a fingerprint reader.
Event Param	None.
Comments	None.

6.2 WFS_EXEE_BIO_SUBJECTDETECTED

Description	This execute event is generated to notify the application when the device has detected a subject in the capture area and an attempt to capture biometric data has been performed.
Event Param	None.
Comments	None.

6.3 WFS_EXEE_BIO_REMOVESUBJECT

Description	This execute event is used to notify an application that the subject should be removed from the capture area of the device.
Event Param	None.
Comments	None.

6.4 WFS_SRVE_BIO_SUBJECTREMOVED

Description	This service event is generated when the subject has been removed from the capture area of the device. This event may be generated at any time.
Event Param	None.
Comments	None.

6.5 WFS_SRVE_BIO_DATACLEARED

Description This mandatory event notifies the application when data has been cleared. This can be the case when the data is cleared automatically after a WFS_CMD_BIO_READ or WFS_CMD_BIO_MATCH command completion, or as a result of an explicit call to the WFS_CMD_BIO_CLEAR or WFS_CMD_BIO_RESET commands.

Input Param LPWFSBIODATACLEARED lpDataCleared;

```
typedef struct _wfs_bio_data_cleared
     {
     DWORD      fwClearData;
     } WFSBIODATACLEARED, *LPWFSBIODATACLEARED;
```

fwClearData
This parameter indicates the data that was cleared from storage as a combination of the following values:

Value	Meaning
WFS_BIO_CLR_SCANNEDDATA	Raw image data that was scanned using the WFS_CMD_BIO_READ command has been cleared.
WFS_BIO_CLR_IMPORTEDDATA	Template data that was imported using the WFS_CMD_BIO_IMPORT command has been cleared.
WFS_BIO_CLR_SETMATCHDATA	Match criteria data that was set using the WFS_CMD_SET_MATCH command has been cleared.

Comments None.

6.6 WFS_EXEE_BIO_ORIENTATION

Description	This event is generated when the biometric subject has an incorrect orientation relative to the device scanner in order to allow an application to prompt a user to correct it.
Event Param	None.
Comments	None.

6.7 WFS_SRVE_BIO_DEVICEPOSITION

Description This service event reports that the device has changed its position status.

Event Param LPWFSBIODEVICEPOSITION lpDevicePosition;

```
typedef struct _wfs_bio_device_position
     {
     WORD                 wPosition;
     } WFSBIODEVICEPOSITION, *LPWFSBIODEVICEPOSITION;
```

wPosition
Position of the device as one of the following values:

Value	Meaning
WFS_BIO_DEVICEINPOSITION	The device is in its normal operating position.
WFS_BIO_DEVICENOTINPOSITION	The device has been removed from its normal operating position.
WFS_BIO_DEVICEPOSUNKNOWN	The position of the device cannot be determined.

Comments None.

6.8 WFS_SRVE_BIO_POWER_SAVE_CHANGE

Description This service event specifies that the power save recovery time has changed.

Event Param LPWFSBIOPOWERSAVECHANGE lpPowerSaveChange;

```
typedef struct _wfs_bio_power_save_change
     {
     USHORT               usPowerSaveRecoveryTime;
     } WFSBIOPOWERSAVECHANGE, *LPWFSBIOPOWERSAVECHANGE;
```

usPowerSaveRecoveryTime
Specifies the actual number of seconds required by the device to resume its normal operational state. This value is zero if the device exited the power saving mode.

Comments If another device class compounded with this device enters into a power saving mode this device will automatically enter into the same power saving mode and this event will be generated.

7. Biometric Device Command Flows – Application Guidelines

The following sections describe the flow of the XFS biometric commands. These application flows are provided as guidelines only.

7.1 Biometric Enrollment Command Flow

The following table describes the flow of enrolling a user using the WFS_CMD_BIO_READ command. Two attempts at scanning are necessary.

Step	Customer	Application	XFS Commands and Events
1.		Begins Enrollment process.	WFS_CMD_BIO_READ *usMode* = WFS_BIO_MODE_SCAN … WFS_EXEE_BIO_PRESENTSUBJECT
2.		Ask customer to present subject to sensor, e.g. finger, eye, palm	
3.	Customer presents subject to biometric sensor.		
4.			WFS_EXEE_BIO_SUBJECTDETECTED event, device scans and collects the biometric data.
5.			WFS_EXEE_BIO_REMOVESUBJECT
6.		Ask customer to remove subject from sensor	
7.	Customer removes subject from biometrics sensor		
8.			WFS_SRVE_BIO_SUBJECTREMOVED
9.			WFS_EXEE_BIO_PRESENTSUBJECT (if another attempt is needed).
10.		Ask customer to present subject to sensor, e.g. finger, eye, palm	
11.	Customer presents subject to biometric sensor.		
12.			WFS_EXEE_BIO_SUBJECTDETECTED event, device scans and collects the biometric data.
13.			WFS_EXEE_BIO_REMOVESUBJECT event
14.			As no further attempts are needed: WFS_CMD_BIO_READ completion
15.		Ask customer to remove subject from sensor.	
16.			WFS_SRVE_BIO_SUBJECTREMOVED
17.		Store biometric data to smart card, database, server/host, etc.	

7.2 Biometric Match Command Flow – Separate Scan and Match

The following table describes the flow of successfully identifying a customer whose biometric template data was previously enrolled and stored on a server/smart card/host system. This template data is first imported using the WFS_CMD_BIO_IMPORT command, which assigns it a unique identifying number. This *usIdentifier* number can then be retrieved using the WFS_INF_BIO_STORAGE_INFO command.

The WFS_CMD_BIO_READ and WFS_CMD_BIO_MATCH commands are then used to scan data and then compare it with the template identified by *usIdentifier*. In this use case the device can perform a separate scan and match operation, therefore the WFS_CMD_BIO_READ command is called to scan the subject's biometric data then the WFS_CMD_BIO_MATCH command is called to perform the match and return the result to the application.

In this case the capability *dwMatchSupported* is reported as WFS_BIO_MTC_STORED_MATCH.

Step	**Customer**	**Application**	**XFS Commands and Events**
1.		Import biometric template data into the device, e.g. from host, smart card, etc.	WFS_CMD_BIO_IMPORT WFS_CMD_BIO_IMPORT completion
2.		Begins scan of customer for matching.	WFS_CMD_BIO_READ *usMode* = WFS_BIO_MODE_MATCH … WFS_EXEE_BIO_PRESENTSUBJECT
3.		Ask customer to present subject to sensor, e.g. finger, eye, palm	
4.	Customer presents subject to biometric sensor.		
5.			WFS_EXEE_BIO_SUBJECTDETECTED event, device scans and stores the customer's biometric data.
6.			WFS_EXEE_BIO_REMOVESUBJECT event
7.			WFS_CMD_BIO_READ completion
8.		Request customer to remove biometric subject from sensor.	
9.			WFS_SRVE_BIO_SUBJECTREMOVED
10.		Application obtains the *usIdentifier* for the imported biometric template data to be matched.	WFS_INF_BIO_STORAGE_INFO
11.		Begin identification process.	WFS_CMD_BIO_MATCH is called with input parameter *usIdentifier* = 12345. The service provider compares the scanned data obtained using the WFS_CMD_BIO_READ command to the previously imported template data identified by *usIdentifier*.
12.			WFS_CMD_BIO_MATCH completion.

7.3 Biometric Match Command Flow – Combined Scan and Match

The following table describes the flow of successfully identifying a customer whose biometric template data was previously enrolled and stored on a server/smart card/host system. This template data is first imported using the WFS_CMD_BIO_IMPORT command, which assigns it a unique identifying number. This *usIdentifier* number can then be retrieved using the WFS_INF_BIO_STORAGE_INFO command.

The WFS_CMD_BIO_READ, WFS_CMD_BIO_SET_MATCH and WFS_CMD_BIO_MATCH commands are then used to scan data and compare it with the template identified by *usIdentifier*. In this use case the device performs a combined scan and match operation, therefore the WFS_CMD_BIO_SET_MATCH command must be used to set the criteria to be used for matching, including the imported template to be identified by *usIdentifier*. When the WFS_CMD_BIO_READ command is then called the device scans the user and performs the comparison as a combined operation. Finally the WFS_CMD_BIO_MATCH command is called to return the result of the comparison to the application.

In this case the capability *dwMatchSupported* is reported as WFS_BIO_MTC_COMBINED_MATCH.

Step	**Customer**	**Application**	**XFS Commands and Events**
1.		Import biometric template data into the device, e.g. from host, smart card, etc.	WFS_CMD_BIO_IMPORT WFS_CMD_BIO_IMPORT completion
2.		Application obtains the *usIdentifier* for an imported biometric template data to be matched.	WFS_INF_BIO_STORAGE_INFO WFS_INF_BIO_STORAGE_INFO completion *usIdentifier* = 12345
3.		Set the criteria to represent what constitutes a successful match, and also the imported template data to be matched.	WFS_CMD_BIO_SET_MATCH is called with input parameter *usIdentifier* = 12345.
4.			WFS_CMD_BIO_SET_MATCH completion
5.		Begins scan of customer for matching.	WFS_CMD_BIO_READ *usMode* = WFS_BIO_MODE_MATCH … WFS_EXEE_BIO_PRESENTSUBJECT
6.		Ask customer to present subject to sensor, e.g. finger, eye, palm	
7.	Customer presents subject to biometric sensor.		
8.			WFS_EXEE_BIO_SUBJECTDETECTED event, device scans and collects the customer's biometric data, then compares it to the previously imported template data identified by *usIdentifier*.
9.			WFS_EXEE_BIO_REMOVESUBJECT event
10.			WFS_CMD_BIO_READ completion
11.		Request customer to remove biometric subject from sensor.	
12.			WFS_SRVE_BIO_SUBJECTREMOVED
13.		Get the result of the comparison.	WFS_CMD_BIO_MATCH is called to return the result of the comparison done at stage 8.
14.			WFS_CMD_BIO_MATCH completion.

7.4 Biometric Scan-Only Command Flow

The following table describes the flow for a simple biometric scanning device which does not support any matching at all. User data is scanned using the WFS_CMD_BIO_READ command but matching is performed externally, for example on a smart card or on a server.

In this case the capability *dwMatchSupported* is reported as WFS_BIO_MTC_NONE.

Step	**Customer**	**Application**	**XFS Commands and Events**
1.		Begin Scanning process.	WFS_CMD_BIO_READ *usMode* = WFS_BIO_MODE_SCAN … WFS_EXEE_BIO_PRESENTSUBJECT
2.		Ask customer to present subject to sensor, e.g. finger, eye, palm	
3.	Customer presents subject to biometric sensor.		
4.			WFS_EXEE_BIO_SUBJECTDETECTED event, device scans and collects the biometric data.
5.			WFS_EXEE_BIO_REMOVESUBJECT event
		Request customer to remove biometric subject from sensor.	
6.			WFS_CMD_BIO_READ completes and returns biometric data.
7.		Request customer to remove biometric subject from sensor.	
8.			WFS_SRVE_BIO_SUBJECTREMOVED
9.		Send biometric data to smart card, database, server/host, etc~~,~~., for matching.	

8. C - Header file

```
/******************************************************************************
*                                                                             *
* xfsbio.h      XFS - Biometrics (BIO) definitions                            *
*                                                                             *
*               Version 3.~~40  (December 6 2019) ~~50  (November 18 2022)
*
*                                                                             *
******************************************************************************/

#ifndef __INC_XFSBIO__H
#define __INC_XFSBIO__H

#ifdef __cplusplus
extern "C" {
#endif

#include <xfsapi.h>

/* be aware of alignment */
#pragma pack (push, 1)



/* values of WFSBIOCAPS.wClass */

#define     WFS_SERVICE_CLASS_BIO                   (17)
#define     WFS_SERVICE_CLASS_NAME_BIO              "BIO"
#define     WFS_SERVICE_CLASS_VERSION_BIO           (~~0x2803~~0x3203) /* Version 3.~~40~~50 */

#define     BIO_SERVICE_OFFSET                      (WFS_SERVICE_CLASS_BIO * 100)

/* BIO Info Commands */

#define     WFS_INF_BIO_STATUS                      (BIO_SERVICE_OFFSET + 1)
#define     WFS_INF_BIO_CAPABILITIES                (BIO_SERVICE_OFFSET + 2)
#define     WFS_INF_BIO_STORAGE_INFO                (BIO_SERVICE_OFFSET + 3)
#define     WFS_INF_BIO_KEY_INFO                    (BIO_SERVICE_OFFSET + 4)

/* BIO Execute Commands */

#define     WFS_CMD_BIO_READ                        (BIO_SERVICE_OFFSET + 1)
#define     WFS_CMD_BIO_IMPORT                      (BIO_SERVICE_OFFSET + 2)
#define     WFS_CMD_BIO_MATCH                       (BIO_SERVICE_OFFSET + 3)
#define     WFS_CMD_BIO_SET_MATCH                   (BIO_SERVICE_OFFSET + 4)
#define     WFS_CMD_BIO_CLEAR                       (BIO_SERVICE_OFFSET + 5)
#define     WFS_CMD_BIO_RESET                       (BIO_SERVICE_OFFSET + 6)
#define     WFS_CMD_BIO_SET_DATA_PERSISTENCE        (BIO_SERVICE_OFFSET + 7)
#define     WFS_CMD_BIO_SET_GUIDANCE_LIGHT          (BIO_SERVICE_OFFSET + 8)
#define     WFS_CMD_BIO_POWER_SAVE_CONTROL          (BIO_SERVICE_OFFSET + 9)
#define     WFS_CMD_BIO_SYNCHRONIZE_COMMAND         (BIO_SERVICE_OFFSET + 10)

/* BIO Events */

#define     WFS_EXEE_BIO_PRESENTSUBJECT             (BIO_SERVICE_OFFSET + 1)
#define     WFS_EXEE_BIO_SUBJECTDETECTED            (BIO_SERVICE_OFFSET + 3)
#define     WFS_EXEE_BIO_REMOVESUBJECT              (BIO_SERVICE_OFFSET + 4)
#define     WFS_SRVE_BIO_SUBJECTREMOVED             (BIO_SERVICE_OFFSET + 5)
#define     WFS_SRVE_BIO_DATACLEARED                (BIO_SERVICE_OFFSET + 6)
#define     WFS_USRE_BIO_ORIENTATION                (BIO_SERVICE_OFFSET + 7)
#define     WFS_SRVE_BIO_DEVICEPOSITION             (BIO_SERVICE_OFFSET + 8)
#define     WFS_SRVE_BIO_POWER_SAVE_CHANGE          (BIO_SERVICE_OFFSET + 9)

/* values of WFSBIOSTATUS.fwDevice */

#define     WFS_BIO_DEVONLINE                      WFS_STAT_DEVONLINE
#define     WFS_BIO_DEVOFFLINE                     WFS_STAT_DEVOFFLINE
#define     WFS_BIO_DEVPOWEROFF                    WFS_STAT_DEVPOWEROFF
```

```
#define     WFS_BIO_DEVNODEVICE                 WFS_STAT_DEVNODEVICE
#define     WFS_BIO_DEVHWERROR                  WFS_STAT_DEVHWERROR
#define     WFS_BIO_DEVUSERERROR                WFS_STAT_DEVUSERERROR
#define     WFS_BIO_DEVBUSY                     WFS_STAT_DEVBUSY
#define     WFS_BIO_DEVFRAUDATTEMPT             WFS_STAT_DEVFRAUDATTEMPT
#define     WFS_BIO_DEVPOTENTIALFRAUD           WFS_STAT_DEVPOTENTIALFRAUD

/* values of WFSBIOSTATUS.dwSubject */

#define     WFS_BIO_SUBJECTPRESENT              (1)
#define     WFS_BIO_SUBJECTNOTPRESENT           (2)
#define     WFS_BIO_SUBJECTUNKNOWN              (3)
#define     WFS_BIO_SUBJECTNOTSUPPORTED         (4)

/* Size and max index of dwGuidLights array */

#define     WFS_BIO_GUIDLIGHTS_SIZE             (32)
#define     WFS_BIO_GUIDLIGHTS_MAX              (WFS_BIO_GUIDLIGHTS_SIZE - 1)

/* Indices of WFSBIOSTATUS.dwGuidLights [...]
              WFSBIOCAPS.dwGuidLights [...] */

#define     WFS_BIO_GUIDANCE_BIO                (0)

/* Values of WFSBIOSTATUS.dwGuidLights [...]
             WFSBIOCAPS.dwGuidLights [...],
             WFSBIOSETGUIDLIGHT.wGuidLight */

#define     WFS_BIO_GUIDANCE_NOT_AVAILABLE      (0x00000000)
#define     WFS_BIO_GUIDANCE_OFF                (0x00000001)
#define     WFS_BIO_GUIDANCE_SLOW_FLASH         (0x00000004)
#define     WFS_BIO_GUIDANCE_MEDIUM_FLASH       (0x00000008)
#define     WFS_BIO_GUIDANCE_QUICK_FLASH        (0x00000010)
#define     WFS_BIO_GUIDANCE_CONTINUOUS         (0x00000080)
#define     WFS_BIO_GUIDANCE_RED                (0x00000100)
#define     WFS_BIO_GUIDANCE_GREEN              (0x00000200)
#define     WFS_BIO_GUIDANCE_YELLOW             (0x00000400)
#define     WFS_BIO_GUIDANCE_BLUE               (0x00000800)
#define     WFS_BIO_GUIDANCE_CYAN               (0x00001000)
#define     WFS_BIO_GUIDANCE_MAGENTA            (0x00002000)
#define     WFS_BIO_GUIDANCE_WHITE              (0x00004000)
#define     WFS_BIO_GUIDANCE_ENTRY              (0x00100000)
#define     WFS_BIO_GUIDANCE_EXIT               (0x00200000)

/* values of WFSBIOSTATUS.wDevicePosition
             WFSBIODEVICEPOSITION.wPosition */

#define     WFS_BIO_DEVICEINPOSITION            (0)
#define     WFS_BIO_DEVICENOTINPOSITION         (1)
#define     WFS_BIO_DEVICEPOSUNKNOWN            (2)
#define     WFS_BIO_DEVICEPOSNOTSUPP            (3)

/* values of WFSBIOSTATUS.wAntiFraudModule */

#define     WFS_BIO_AFMNOTSUPP                  (0)
#define     WFS_BIO_AFMOK                       (1)
#define     WFS_BIO_AFMINOP                     (2)
#define     WFS_BIO_AFMDEVICEDETECTED           (3)
#define     WFS_BIO_AFMUNKNOWN                  (4)


/* values of WFSBIOCAPS.fwType */

#define     WFS_BIO_TYPE_FACIAL_FEATURES        (0x0001)
#define     WFS_BIO_TYPE_VOICE                  (0x0002)
#define     WFS_BIO_TYPE_FINGERPRINT            (0x0004)
#define     WFS_BIO_TYPE_FINGERVEIN             (0x0008)
#define     WFS_BIO_TYPE_IRIS                   (0x0010)
#define     WFS_BIO_TYPE_RETINA                 (0x0020)
#define     WFS_BIO_TYPE_HAND_GEOMETRY          (0x0040)
```

```
#define     WFS_BIO_TYPE_THERMAL_FACE           (0x0080)
#define     WFS_BIO_TYPE_THERMAL_HAND           (0x0100)
#define     WFS_BIO_TYPE_PALM_VEIN              (0x0200)
#define     WFS_BIO_TYPE_SIGNATURE              (0x0400)

/* values of WFSBIOCAPS.fwDataFormats and
             WFSBIODATATYPE.dwFormat and
             WFSBIOREAD.lpdwDataFormats */

#define     WFS_BIO_ISOFID                      (0x0001)
#define     WFS_BIO_ISOFMD                      (0x0002)
#define     WFS_BIO_ANSIFID                     (0x0004)
#define     WFS_BIO_ANSIFMD                     (0x0008)
#define     WFS_BIO_QSO                         (0x0010)
#define     WFS_BIO_WSQ                         (0x0020)
#define     WFS_BIO_RESERVED_RAW_1              (0x0040)
#define     WFS_BIO_RESERVED_TEMPLATE_1         (0x0080)
#define     WFS_BIO_RESERVED_RAW_2              (0x0100)
#define     WFS_BIO_RESERVED_TEMPLATE_2         (0x0200)
#define     WFS_BIO_RESERVED_RAW_3              (0x0400)
#define     WFS_BIO_RESERVED_TEMPLATE_3         (0x0800)

/* values of WFSBIOCAPS.fwEncryptionAlgorithms and
             WFSBIODATATYPE.dwAlgorithm */

#define     WFS_BIO_CRYPT_NONE                   (0x0000)
#define     WFS_BIO_CRYPT_TRIDESECB              (0x0001)
#define     WFS_BIO_CRYPT_TRIDESCBC              (0x0002)
#define     WFS_BIO_CRYPT_TRIDESCFB              (0x0004)
#define     WFS_BIO_CRYPT_RSA                    (0x0008)

/* values of WFSBIOCAPS.fwStorage */

#define     WFS_BIO_STORAGE_NONE                 (0x0000)
#define     WFS_BIO_STORAGE_SECURE               (0x0001)
#define     WFS_BIO_STORAGE_CLEAR                (0x0002)

/* values of WFSBIOCAPS.fwPersistenceModes and
             WFSBIOSTATUS.dwDataPersistence and
             WFSBIOPERSISTDATA.dwPersistenceMode */

#define     WFS_BIO_PS_NONE                     (0x0000)
#define     WFS_BIO_PS_PERSIST                  (0x0001)
#define     WFS_BIO_PS_AUTOCLEAR                (0x0002)

/* values of WFSBIOCAPS.dwMatchSupported */

#define     WFS_BIO_MTC_STORED_MATCH_NONE       (0x0000)
#define     WFS_BIO_MTC_STORED_MATCH            (0x0001)
#define     WFS_BIO_MTC_COMBINED_MATCH          (0x0002)

/* values of WFSBIOCAPS.fwScanModes and
             WFSBIOREAD.usMode */

#define     WFS_BIO_MODE_SCAN                   (0x0001)
#define     WFS_BIO_MODE_MATCH                  (0x0002)

/* values of WFSBIOCAPS.fwCompareModes and
             WFSBIOMATCH.usCompareMode */

#define     WFS_BIO_COMP_NONE                   (0x0000)
#define     WFS_BIO_COMP_VERIFY                 (0x0001)
#define     WFS_BIO_COMP_IDENTIFY               (0x0002)

/* values of WFSBIOCAPS.fwClearData and
             WFSBIOCLEAR.fwClearData and
             WFSBIORESET.fwClearData and
             WFSBIODATACLEARED.fwClearData */

#define     WFS_BIO_CLR_NONE                    (0x0000)
```

```
#define     WFS_BIO_CLR_SCANNEDDATA             (0x0001)
#define     WFS_BIO_CLR_IMPORTEDDATA            (0x0002)
#define     WFS_BIO_CLR_SETMATCHDATA            (0x0004)

/* values of WFSBIOKEYINFO.dwUse */

#define     WFS_BIO_USECRYPT                    (0x0001)
#define     WFS_BIO_USERSAPUBLIC                (0x0002)

/* XFS BIO Errors */

#define WFS_ERR_BIO_NOIMPORTEDDATA              (-(BIO_SERVICE_OFFSET + 0))
#define WFS_ERR_BIO_READFAILED                  (-(BIO_SERVICE_OFFSET + 1))
#define WFS_ERR_BIO_MODENOTSUPP                 (-(BIO_SERVICE_OFFSET + 2))
#define WFS_ERR_BIO_FORMATNOTSUPP               (-(BIO_SERVICE_OFFSET + 3))
#define WFS_ERR_BIO_INVALIDDATA                 (-(BIO_SERVICE_OFFSET + 4))
#define WFS_ERR_BIO_CAPACITYEXCEEDED            (-(BIO_SERVICE_OFFSET + 5))
#define WFS_ERR_BIO_INVALIDIDENTIFIER           (-(BIO_SERVICE_OFFSET + 6))
#define WFS_ERR_BIO_NOCAPTUREDDATA              (-(BIO_SERVICE_OFFSET + 7))
#define WFS_ERR_BIO_KEYNOTFOUND                 (-(BIO_SERVICE_OFFSET + 8))
#define WFS_ERR_BIO_INVALID_PORT                (-(BIO_SERVICE_OFFSET + 9))
#define WFS_ERR_BIO_POWERSAVETOOSHORT           (-(BIO_SERVICE_OFFSET + 10))
#define WFS_ERR_BIO_COMMANDUNSUPP               (-(BIO_SERVICE_OFFSET + 11))
#define WFS_ERR_BIO_SYNCHRONIZEUNSUPP           (-(BIO_SERVICE_OFFSET + 12))
#define WFS_ERR_BIO_INVALIDCOMPAREMODE          (-(BIO_SERVICE_OFFSET + 13))
#define WFS_ERR_BIO_INVALIDTHRESHOLD            (-(BIO_SERVICE_OFFSET + 14))



/*=================================================================*/
/* BIO Info Command Structures and variables */
/*=================================================================*/

typedef struct _wfs_bio_status
{
    WORD                    fwDevice;
    DWORD                   dwSubject;
    BOOL                    bCaptured;
    DWORD                   dwDataPersistence;
    DWORD                   dwRemainingStorage;
    LPSTR                   lpszExtra;
    WORD                    wDevicePosition;
    DWORD                   dwGuidLights[WFS_BIO_GUIDLIGHTS_SIZE];
    USHORT                  usPowerSaveRecoveryTime;
    WORD                    wAntiFraudModule;
} WFSBIOSTATUS, *LPWFSBIOSTATUS;

typedef struct _wfs_bio_caps
{
    WORD                    wClass;
    DWORD                   fwType;
    BOOL                    bCompound;
    USHORT                  usMaxCapture;
    DWORD                   dwTemplateStorage;
    DWORD                   fwDataFormats;
    DWORD                   fwEncryptionAlgorithms;
    WORD                    fwStorage;
    DWORD                   fwPersistenceModes;
    DWORD                   dwMatchSupported;
    WORD                    fwScanModes;
    WORD                    fwCompareModes;
    DWORD                   fwClearData;
    LPSTR                   lpszExtra;
    DWORD                   dwGuidLights[WFS_BIO_GUIDLIGHTS_SIZE];
    BOOL                    bPowerSaveControl;
    BOOL                    bAntiFraudModule;
    LPDWORD                 lpdwSynchronizableCommands;
} WFSBIOCAPS, *LPWFSBIOCAPS;

typedef struct _wfs_bio_data_type
```

```
{
    DWORD                    dwFormat;
    DWORD                    dwAlgorithm;
    LPSTR                    lpszKeyName;
} WFSBIODATATYPE, *LPWFSBIODATATYPE;

typedef struct _wfs_bio_storage
{
    USHORT                   usIdentifier;
    LPWFSBIODATATYPE         lpType;
} WFSBIOSTORAGE, *LPWFSBIOSTORAGE;

typedef struct _wfs_bio_storage_list
{
    USHORT                   usCount;
    LPWFSBIOSTORAGE         *lppStorageList;
} WFSBIOSTORAGELIST, *LPWFSBIOSTORAGELIST;

typedef struct _wfs_bio_key_info
{
    LPSTR                    lpszKeyName;
    DWORD                    dwUse;
    BOOL                     bLoaded;
} WFSBIOKEYINFO, *LPWFSBIOKEYINFO;

/*==================================================================*/
/* BIO Execute Command Structures */
/*==================================================================*/

typedef struct _wfs_bio_read
{
    USHORT                   usCount;
    LPWFSBIODATATYPE         *lppTypes;
    USHORT                   usNumCaptures;
    USHORT                   usMode;
} WFSBIOREAD, *LPWFSBIOREAD;

typedef struct _wfs_bio_hex_data
{
    USHORT                   usLength;
    LPBYTE                   lpbData;
} WFSXBIODATA, *LPWFSXBIODATA;

typedef struct _wfs_bio_data
{
    LPWFSBIODATATYPE         lpType;
    LPWFSXBIODATA            lpxData;
} WFSBIODATA, *LPWFSBIODATA;

typedef struct _wfs_bio_read_data
{
    USHORT                   usCount;
    LPWFSBIODATA            *lppBioDataList;
} WFSBIOREADDATA, *LPWFSBIOREADDATA;

typedef struct _wfs_bio_import_data
{
    USHORT                   usCount;
    LPWFSBIODATA            *lppBioDataList;
} WFSBIOIMPORTDATA, *LPWFSBIOIMPORTDATA;

typedef struct _wfs_bio_match
{
    USHORT                   usCompareMode;
    USHORT                   usIdentifier;
    USHORT                   usMaximum;
    USHORT                   usThreshold;
} WFSBIOMATCH, *LPWFSWFSBIOMATCH;

typedef struct _wfs_bio_candidate
```

```
{
    USHORT                   usConfidenceLevel;
    USHORT                   usIdentifier;
    LPWFSBIODATA             lpData;
} WFSBIOCANDIDATE, *LPWFSBIOCANDIDATE;

typedef struct _wfs_bio_match_result
{
    USHORT                   usCount;
    LPWFSBIOCANDIDATE       *lppTemplateList;
} WFSBIOMATCHRESULT, *LPWFSBIOMATCHRESULT;

typedef struct _wfs_bio_clear
{
    DWORD                     fwClearData;
} WFSBIOCLEAR, *LPWFSBIOCLEAR;

typedef struct _wfs_bio_reset
{
    DWORD                     fwClearData;
} WFSBIORESET, *LPWFSBIORESET;

typedef struct _wfs_bio_persist_data
{
    DWORD                    dwPersistenceMode;
} WFSBIOPERSISTDATA, *LPWFSBIOPERSISTDATA;

typedef struct _wfs_bio_set_guidlight
{
    WORD                     wGuidLight;
    DWORD                    dwCommand;
} WFSBIOSETGUIDLIGHT, *LPWFSBIOSETGUIDLIGHT;

typedef struct _wfs_bio_power_save_control
{
    USHORT                   usMaxPowerSaveRecoveryTime;
} WFSBIOPOWERSAVECONTROL, *LPWFSBIOPOWERSAVECONTROL;

typedef struct _wfs_bio_synchronize_command
{
    DWORD                    dwCommand;
    LPVOID                   lpCmdData;
} WFSBIOSYNCHRONIZECOMMAND, *LPWFSBIOSYNCHRONIZECOMMAND;


/*=================================================================*/
/* BIO Events Structures */
/*=================================================================*/

typedef struct _wfs_bio_data_cleared
{
    DWORD                     fwClearData;
} WFSBIODATACLEARED, *LPWFSBIODATACLEARED;

typedef struct _wfs_bio_device_position
{
    WORD                     wPosition;
} WFSBIODEVICEPOSITION, *LPWFSBIODEVICEPOSITION;

typedef struct _wfs_bio_power_save_change
{
    USHORT                   usPowerSaveRecoveryTime;
} WFSBIOPOWERSAVECHANGE, *LPWFSBIOPOWERSAVECHANGE;



/* restore alignment */
#pragma pack (pop)

#ifdef __cplusplus
```

```
}       /*extern "C"*/
#endif

#endif  /* __INC_XFSBIO__H */
```