Specification

BSAPI Reference Manual

Table of Contents

1 Introduction

1.1 Terminology

- **BSAPI** – Biometric Services API
- **FM** – fingerprint module (fingerprint reader device) connected to host
- **HOST** – computer where the FM is connected to
- **BioAPI** – the industry standard for biometric API, developed by BioAPI Consortium [www.bioapi.org]. All references to BioAPI in this document assume version 1.1 of the standard.
- **BioAPI Framework** – reference implementation of BioAPI, by BioAPI Consortium.
- **(BioAPI) BSP** – Biometry Service Provider; 3rd party module pluggable into BioAPI Framework. BSP provides support for particular device (e.g. fingerprint reader).
- **BSP API** – API below BioAPI Framework, specifying interface between BioAPI framework and BSPs.
- **TFMESSBSP** – Upek BSP implementation, supporting TFM and ESS devices.

1.2 Preview

The purpose of this document is to describe architecture and interface of Biometric Services API (BSAPI).

BSAPI was designed to meet these requirements:

- Similar to BioAPI: development team familiar with BioAPI should be able to adopt BSAPI easily and in reasonable short time.
- Compatibility with TFMESSBSP: BSAPI will provide functions which can behave exactly as corresponding functions in TFMESSBSP. (Future version of TFMESBSP can be implemented on top of BSAPI easily.)
- BSAPI will be implemented in simple DLL. Unlike BioAPI, BSAPI does not require any framework to use it.
- BSAPI will provide more functionality comparing to BioAPI, so more biometric and miscellaneous features of our devices will be available (e.g. navigation).
- Avoid need to use any low level API (e.g. PTAPI).

Do not misunderstand the term “compatibility” with BioAPI as used in the list above. There will be no binary compatibility between BioAPI and BSAPI, nor compatibility on source level. BSAPI will provide set of function which will allow all code using BioAPI (with TFMESSBSP) to be rewritten to use BSAPI instead, with exactly same behavior.

In fact, as further described below, future versions of TFMESSBSP will be thin layer on top of BSAPI.

1.3 Architecture

BSAPI is composed from several dynamic libraries (.DLL on Windows. On other system, other file extension is used.):

- **BSAPI.DLL**, which contains all core functionality.
- **BSGUI.DLL** providing default implementation of GUI callback. The library loads graphics from separated .zip file (window decorations and feedback images), so that customization of look&feel is possible. The

library supports multiple languages (localization), generally those supported in our other software. Applications can provide their own callback and in such case, they do not use this library. Please note that currently BSGUI.DLL is available only for Windows platform.

- **BSSRV.DLL** is a library with biometric functions which do **not** require a biometric sensor device present. Intended use case is server in applications with client-server architecture, where server side manages a database of fingerprint templates acquired by clients, which are assumed to use BSAPI.DLL.

Note that BSSRV.DLL is provided only within full BSAPI SDK package. It is not provided in the lite version of the SDK.

Application developers can choose from several approaches how to deal with the GUI:

- Use BSGUI.DLL as it is, which will guarantee appearance of the applications, consistent with UPEK applications.
- Use BSGUI.DLL, with customized .zip file: some or all of the graphic files in the .zip file can be modified or replaced with other graphics.
- Implement your own callback implementation. This allows maximal freedom in customization, including adding support for languages not supported by BSGUI.DLL.

1.4 Naming Conventions

All identifiers (names of constants, types and functions) will use prefix “ABS”, and follow the patterns below:

- ABS_MACRO_IDENTIFIER
- ABS_TYPE_IDENTIFIER
- ABSFunctionIdentifier()

Server-side functions (BSSRV.DLL) use prefix ABSSrv:

- ABSSrvFunctionIdentifier()

BSSRV.DLL shares definition of constants and types with BSAPI.DLL.

In function prototypes special words IN, OUT and INOUT are used, to denote if the parameter is used to pass data into the function, return some result data or both.

2 BSAPI.DLL Functions

2.1 General Description

BSAPI.DLL is main library if BSAPI SDK. It provides a set of functions which can read data from supported fingerprint sensor devices, and which apply various biometric algorithms to these data.

The main header file declaring functions of BSAPI is bsapi.h. The header includes bstypes.h and bserror.h which declare types and error status codes. The latter two headers are shared with the other libraries the SDK consists of.

2.1.1 Supported FM Devices

BSAPI.DLL supports these devices:

- Chipset devices with strip sensors based on TCD41, TCD42, TCD50v1, TCD50v2.
- Chipset devices with area sensors based on TCD21 (TFM Module 2.0), TCD50v3.
- SONLY devices with strip sensors based on TCS4B, TCS4C, TCS5B, TCS4K.
- SONLY devices with area sensors based on ST9, Cypress. Note that currently this category of devices is supported only on Windows.

On Windows, BSAPI.DLL supports also biometry-enabled composite devices manufactured by 3rd parties, if these conditions are met:

- The composite device has UPEK sensor embedded, which is supported by BSAPI.
- The manufacturer of the composite device provides original UPEK driver, which is modified for use with the composite device (e.g. it is registered for VID and PID of the composite device).

If such driver is installed, BSAPI "sees" such devices in enumeration with ABSEnumerate and can open the device with ABSOpen as any normal UPEK device.

If you develop application supposed to work only with the composite device, you can filter the set of devices the BSAPI sees by specifying VID and PID in DSN string parameter of ABSEnumerate and ABSOpen.

2.1.2 Error Handling

Almost all BSAPI.DLL functions return a status code **ABS_STATUS**. Code **ABS_STATUS_OK** (zero) means success. All other values denote an error condition.

You may call ABSGetLastErrorInfo to retrieve more information about the error condition. Note that the information is intended as a help for application and library developers and it's not intended to be presented to end users.

If any BSAPI.DLL function fails, it frees any resources it might allocate. Values of output parameters are defined only if the function succeeds i.e. if it returns ABS_STATUS_OK.

2.1.3 Memory Management

Some BSAPI.DLL functions allocate memory returned via output parameter to the calling application. The application must use function ABSFree to free memory allocated by BSAPI.DLL in these cases.

Note the output parameters have a defined value only when the function returns ABS_STATUS_OK. I.e. in a case the BSAPI function fails, the output parameters have undefined values and no memory is allocated for output parameters hence you should not call ABSFree.

2.1.4 Interactive Operations

Some of the BSAPI.DLL functions expect users interaction with FM. All these functions are collectively called **interactive operations** in this document. Interactive operation functions share the way how the interaction is achieved.

All those function have pointer to structure ABS_OPERATION as their second parameter (just after handle of a session). When you call any interactive operation function, it blocks until the operation finishes or until it is canceled. While the operation is processing, callback specified by ABS_OPERATION is repeatedly called so that the application can provide feedback to the end user.

Note that the callback implementation has some limitations. The behavior is not defined if you don't respect them:

- You cannot throw exceptions from the callback (if you use BSAPI from C++ or other language which supports them).
- You cannot call majority of BSAPI functions from the callback. You can safely call only ABSFree, ABSCancelOperation and ABSGetLastErrorInfo from the callback.

Since version 3.5 of BSAPI, the callback is always called from a thread context where the interactive operation function has been called. (In older versions of BSAPI, this was not guaranteed).

You may cancel any running interactive operation with ABSCancelOperation if needed. You may call this function either from the callback itself or from any other thread if you associated unique operation ID to the operation you need to cancel.

The operation ID serves exactly this purpose: it names the operation so that it can be canceled from another thread. When using this approach, the application developer has to use a consistent policy in his code to manage the operation IDs. The application cannot start multiple concurrent interactive operations with the same ID (with exception of zero, which creates an anonymous operation which cannot be canceled from another thread). Any attempt to start new interactive operation with ID already used in another thread will result in error ABS_STATUS_INVALID_PARAMETER.

Note the structure ABS_OPERATION contains member Flags which can further influence how the callback is called.

2.1.5 Multithreading

In general, BSAPI.DLL is thread-safe. You can call BSAPI.DLL functions concurrently from multiple threads, including multiple biometric operations on one FM. BSAPI automatically schedules the operations communicating with the FM. See also Scheduling of Biometric Operations.

The only exception are functions ABSInitialize, ABSInitializeEx and ABSTerminate. These three functions are **not** thread-safe. This is usually not a problem, because they are called as part of application initialization and termination respectively.

2.1.6 Scheduling of Biometric Operations

BSAPI automatically schedules operations which need to communicate with single FM. When multiple such operations are run concurrently from multiple threads, BSAPI automatically suspends one of the operations until the other operation stops the communication with the FM.

When interactive operation is being suspended, its corresponding callback function receives message ABS_MSG_PROCESS_SUSPEND and then, after the operation is being resumed, message ABS_MSG_PROCESS_RESUME.

Short-term non-interactive functions cummunicating with FM (e.g. ABSSetAppData or ABSSetLED) have the highest priority, i.e. calling them will temporarily suspend any ongoing interactive operation.

Navigation (ABSNavigate) has the lowest priority, so the navigation is suspended by any other operation communicating with FM.

2.1.7 Anti-latent Checking

In general the goal of the anti-latent check is to minimize the negative impact of residual fingerprint left on the surface of area sensor. This negative impact has two forms – security (risk of a false accept) and convenience (risk of false reject due to the lowered image quality).

Note that for strip sensors, such checking is never performed because there is no such danger, so for these sensors the API automatically reports the last scan as not being latent.

You can manually perform the anti-latent checking by calling function ABSCheckLatent. The function is able to perform two basic operations:

- Check operation: The most recent finger scan (in a context of session with fingerprint device) is compared with the data of last stored scan in the device's memory.
- Store operation: Store the most recent scan to device's memory.

Caller can specify the desired operation using flags ABS_LATENT_OP_CHECK and/or ABS_LATENT_OP_STORE respectivelly. If both of the two flags are specified (or when none is specified, as it is considered the default behavior), the function performs combination of both operations. As a first step the check operation is performed, and if the most recent scan was not detected as latent one, it is then stored for any subsequent anti-latent checking.

Aside from the function ABSCheckLatent, there is also an implicit anti-latent checking built-in into ABSEnroll and ABSVerify operations. They both behave use both: the check operation as well as the store operation. This implicit anti-latent checking can be disabled by setting global parameter ABS_PARAM_LATENT_CHECK with ABSSetGlobalParameter.

As you can see, you should never perform the check operation just after store operation, as then the most recent swipe will be evaluated as a latent one. The reason is quite obvious: the most recent swipe is compared with copy of the same data. As a follow up, you should never call ABSCheckLatent just after ABSEnroll or ABSVerify (unless you have disabled the implicit anti-latent checking).

2.1.8 Power Control

For some end users, especialy those using mobile deiveces as notebooks or netbooks, power consumption is an important aspect. BSAPI allows applications to control power consumtption of fingerprint devices which support it.

In general, UPEK devices support these three modes of power control:

- **Full-power mode:** The device is full powered and immediately ready for operation.
- **Half-power mode:** The device as a whole is powered, but is firmware/software decided to power-off sensor in the device (the sensor is the highest power consumer). When needed, the device automatically reenables poweringthe sensor when it detects that user places a finger on it (HW finger detection).
- **Low-power mode:** The devices allows the system to put it into selective suspend. Note BSAPI supports this mode only on Windows platforms currently.

BSAPI has several global parameters which control how BSAPI manages the power consumption of the device. By default (when ABS_PARAM_POWER_SAVE_MODE is set to 2), BSAPI tracks user's activity and keeps full-power mode only until timeout ABS_PARAM_POWER_SAVE_ACTIVE_TO_SLEEP_TIMEOUT elapses and then switches the device to hal-power-mode. If there is still no user activity until ABS_PARAM_POWER_SAVE_TIMEOUT occures, the device is switched to low-power regime.

Optionally (by setting ABS_PARAM_POWER_SAVE_RESET_ON_OPERATION_START), the power saving timeouts can be reset whenever an interractive operation starts. I.e. when enabled, the operation then always starts in full-power mode.

Application can explicitely enable/disable low-power mode by setting ABS_PARAM_POWER_SAVE_MODE to 0 or 1. Furthermore functions ABSRawGrab and ABSRawGrabImage allow setting of the power mode for duration of the function call via its pProfileData parameter.

With global parameter ABS_PARAM_POWER_SAVE_CHECK_KEYBOARD, the application can control whether by the "user activity" only activity on fingerprint sensor is understood, or activity on keyboard and mouse as well (Windows only).

2.1.9 Windows Service Compatibility Mode

On Windows, BSAPI.DLL can be initialized in a Windows service compatible mode. To achieve this, call ABSInitalizeEx with the flag ABS_INIT_FLAG_NT_SERVICE. Using BSAPI.DLL in a context of service without this step leads to an undefined behaviour.

Keep in mind that the service compatible mode has some limitations:

- BSAPI does not allow to open remote sessions via Citrix or Terminal Services. See chapter Support for Terminal Services and Citrix for details.
- BSAPI then never treats keyboard or mouse as an user activity with-in the power control context, regardless ABS_PARAM_POWER_SAVE_CHECK_KEYBOARD value. See chapter Power Control for details.

2.1.10 Support for Terminal Services and Citrix

This section applies only for Windows platform.

Since version 3.5, BSAPI supports opening fingerprint sensor devices on remote sessions (via Terminal Services or Citrix). This means that user can work remotely and use a sensor connected to the client computer (i.e. where the user is really sitting) with the application running on the remote computer (server).

Note however that the Citrix and Terminal Services clients required for thsi feature to work are available only in full BSAPI SDK package. The clients are not provided within BSAPI Lite SDK.

2.1.10.1 Application Developer's Point of View

In context of Terminal Services and Citrix, there are three modes of BSAPI operation. Developer determines the mode by using two flags of ABSInitializeEx function: ABS_INIT_FLAG_FORCE_LOCAL_SENSOR and ABS_INIT_FLAG_FORCE_REMOTE_SENSOR. For an obvious reason, these flags cannot be used together.

When none of the flags is set (or when using ABSInitialize), an automatic mode is used. In this mode, local fingerprint devices are used when the application is not running in a remote session; and remote devices when the application is running in a remote session (either Terminal Services or Citrix). When the session state changes (e.g. user disconnects from the remote session of Terminal Services and moves to a computer where he continues to work with the application locally), all subsequent calls to BSAPI.DLL working with the fingerprint device will fail with ABS_STATUS_REMOTE_COMM_ERROR.

When ABS_INIT_FLAG_FORCE_LOCAL_SENSOR is used, BSAPI never opens a remote fingerprint device and will always work only with the devices connected locally to the computer where the application is running.

When ABS_INIT_FLAG_FORCE_REMOTE_SENSOR is used, BSAPI will always work only with remote devices (on the client side). When the application is not actually running in a remote session, function ABSOpen will always fail with ABS_STATUS_REMOTE_COMM_ERROR.

Note that usage of remote fingerprint devices in a NT service compatible mode is not supported. Hence when flag ABS_INIT_FLAG_NT_SERVICE is used with ABSInitializeEx, then automatic mode is equivalent to ABS_INIT_FLAG_FORCE_LOCAL_SENSOR; and ABS_INIT_FLAG_FORCE_REMOTE_SENSOR cannot be used at all.

2.1.10.2 Remote Sessions and Error Handling

In case of problems with communication to the remote sensor, communication always ends with ABS_STATUS_REMOTE_COMM_ERROR status code.

This includes the following situations:

- BSAPI.DLL was initialized in the default automatic mode and state of the OS session has changed. E.g. the user was using the BSAPI application locally and after some time he moved himself to different computer and connected remotely via Terminal Services to the same session. Since that point any subsequent calls into BSAPI will fail with ABS_STATUS_REMOTE_COMM_ERROR as (according to the automatic mode), the BSAPI now rejects to use the local sensor, until the application reopens BSAPI session with any device connected to the client computer.
- BSAPI.DLL was initialized with ABS_INIT_FLAG_FORCE_REMOTE_SENSOR, but the application is not actually running in the remote session context (neither Terminal Services, nor Citrix).
- Some (temporary or permanent) network communication problem occurs.

Applications typically response to this error by closing the actual BSAPI session with ABSClose and optionally attempting to open new BSAPI session with ABSOpen. This function may still fail with this error code if the error condition persists. The application cannot assume it will be able to open exactly the same device.

Note that when working only with local sensors (ABS_INIT_FLAG_FORCE_LOCAL_SENSOR), this error never occurs. Even when the application is running in the remote session context, it will work strictly with devices connected locally (i.e. to the computer where the application is running).

When BSAPI attempts to use remote fingerprint devices (either in automatic mode, or by using ABS_INIT_FLAG_FORCE_REMOTE_SENSOR flag), and the UPEK client is not installed on the client side, a call to ABSOpen will fail with ABS_STATUS_NOT_SUPPORTED.

2.1.10.3 Supported Environments

Terminal Services:

- Client side: Windows XP, Windows Vista, Windows 7
- Server side: Windows XP, Windows Vista, Windows Server 2003, Windows Server 2008, Windows Server 2008R2

Citrix:

- Client side: Windows XP, Windows Vista, Windows 7; with Citrix XenApp plugin 11
- Server side: Windows Server 2003 or Windows Server 2008; with Citrix XenApp server 4.5

2.1.10.4 Setup Instructions

Terminal Services:

1. On the server, enable the Terminal Services support in the system. Right-click My Computer, select Properties. A dialog is displayed. Select the Remote tab and check the "Allow users to connect remotely to this computer" checkbox .

2. On the client, install the UPEK Terminal Services Client (tsclient.msi). You should **not** be connected to the remote desktop at the time of the installation. Note you have to install a 32-bit version of the client on a 32-bit system, and a 64-bit version of the client on a 64-bit system.

3. On the client, install the fingerprint device driver and plug the fingerprint device into the computer.

4. On the client, start a Microsoft Remote Desktop client application (you can usually find it in Start >> Programs >> Accessories). Fill in the name of the computer where the BSAPI application is installed (server) and connect to it.

5. In the remote desktop, start and use a BSAPI-based application installed on the server. The application will use the fingerprint device connected to your client computer (unless the application disabled that with the flag ABS_INIT_FLAG_FORCE_LOCAL_SENSOR).

Citrix:

1. On the server, the Citrix XenApp server must be installed.

2. On the client, the Citrix XenApp plugin must be installed.

3. On the client, install the UPEK Citrix Client (tscitrix.msi). You should **not** be connected to the remote desktop at the time of the installation. Note there is only a 32-bit version of the client, which works both in 32-bit and 64-bit systems.

4. On the client, install the fingerprint device driver and plug the fingerprint device into the computer.

5. From the client, connect to the server using the Citrix XenApp plugin.

6. In the remote desktop, start and use a BSAPI-based application installed on the server. The application will use the fingerprint device connected to your client computer (unless the application disabled that with the flag ABS_INIT_FLAG_FORCE_LOCAL_SENSOR).

2.1.11 Blinking with LEDs

Some UPEK devices are equipped with one or more LEDs. Their blinking can be part of friendly interaction with users. BSAPI provides functions to control the blinking of those LEDs. Please note that some devices are also equipped with LEDs controlled directly by the device itself (HW). Such LEDs cannot be controlled with BSAPI.

Note however that level of support varies a lot among different HW devices.

Actually there are two pairs of functions controlling the LEDs.

Older ABSSetLED() and ABSGetLED() are easier to use, but allow to work properly only with some (simpler) blinking modes. Newer ABSSetLedEx() and ABSGetLedEx() offer more versability.

It is recommanded to not mix their use in a single application. I.e. you should choose whether you rather use the older ABSSetLED() and ABSGetLED(), or the newer counterparts.

New application should prefer the newer and more powerful functions ABSSetLedEx() and ABSGetLedEx().

There are several blinking modes supported.

Also remember that when some devices enter a deep sleep or standby, the LEDs will be turned off for the duration of the sleep. After wakeup the LEDs will resume the status they had before the sleep (including the blinking).

2.1.11.1 Mode ABS_LED_MODE_AUTO

This mode is set by default after ABSOpen(). In this mode the blionking of LEDs is completely controlled by BSAPI. Exact blinking behavior can differ in various BSAPI versions, and it is also device-dependent.

There are no additional parameters describing this mode. In case of ABSSetLED(), you sghould set dwLED1 and dwLED2 to zero. With ABSSetLedEx() the parameter pLedParam should be set to NULL or point to empty ABS_DATA.

2.1.11.2 Mode ABS_LED_MODE_MANUAL

This mode allows to control up to two LEDs installed on the device manually by the application.

Behavior of each of the two LEDs is described by one doubleword value (ABS_DWORD). Bits of the value specify blinking pattern and duration of the state described by each bit of the pattern:

- **Bits 0-15:** Blinking pattern. The device iterates over the bits (one bit in one clock tick as determined by bits 16-19, see below) and turns the LED on (bit value is 1) or off (bit value is 0). Bit 0 is displayed first, Bit 1 in the next clock tick etc.
- **Bits 16-19:** Blinking clock duration exponent. Value 0 stops the blinking, value 1 = 1 msec per a pattern bit, value 2 = 2 msec per a pattern bit, value 3 = 4 msec per a pattern bit, … value 15 = 16384 sec per a pattern bit (value N = 2^{N-1} msec per a pattern bit).

I.e. to turn a LED permanently ON set all bits to one. To switch a LED permanently OFF reset all bits to zero.

In case of the older functions ABSSetLED() and ABSGetLED(), the doublewords describing the LED behavior correspond to paramaters dwLED1 and dwLED2, or pdwLED1 and pdwLED2 respectivelly.

Newer functions ABSSetLedEx() and ABSGetLedEx() use members Led1 and Led2 of the structure ABS_LED_PARAMS_MANUAL, wrapped in ABS_DATA via the parameter pLedParams or ppLedParams respectivelly.

Note specific for SONLY devices: The blinking pattern can only be one of these values: 0x0000, 0x0001, 0x0003, 0x007, 0x000f, 0x001f, 0x003f, 0x007f, 0x00ff, 0x01ff, 0x03ff, 0x07ff, 0x0fff, 0x1fff, 0x3fff, 0x7fff and 0xffff. The value of period duration exponent can be only 0, 1, 2, 3, 4, 5, 6, 7, 8 and 15 for SONLY devices. All other values are not supported by SONLY devices and using them will lead to error ABS_STATUS_INVALID_PARAMETER.

2.1.11.3 Mode ABS_LED_MODE_MANUAL2

This advanced manual mode allows very detailed settings of up to three LEDs. The behavior is described by the structure ABS_LED_PARAMS_MANUAL2.

Note that this mode is only supported by TCS5D devices. Attempt to use this mode with other device will fail.

Older function ABSSetLED() cannot be used to set this mode. Furthermore if this mode is set, then calling ABSGetLED() only sets pdwMode to indicate this mode is active, but the paramaters are not retrieved to the caller. If your application uses this mode, you should not use these functions.

Newer functions ABSSetLedEx() and ABSGetLedEx() use the structure ABS_LED_PARAMS_MANUAL2, wrapped in ABS_DATA via the parameter pLedParams or ppLedParams respectivelly.

2.1.11.4 Mode ABS_LED_MODE_OFF

This mode just turns all LEDs (controlled from SW) permanently off.

There are no additional parameters describing this mode. In case of ABSSetLED(), you sghould set dwLED1 and dwLED2 to zero. With ABSSetLedEx() the parameter pLedParam should be set to NULL or point to empty ABS_DATA.

2.2 Application General Functions

The Application General Functions allow to initialize the BSAPI library, open and close logical connections to FM and perform other miscellaneous tasks.

2.2.1 ABSInitialize

```
ABS_STATUS ABSInitialize(
     void
)
```

Description	Initialize the BSAPI.DLL library. BSAPI.DLL must be initialized before you can call any other function. It is called typically during application startup. Note that when using BSAPI.DLL in a context of Windows NT service, it cannot be initialized with this function. Instead use ABSInitializeEx with flags ABS_INIT_FLAG_NT_SERVICE. Calling ABSInitialize is equivalent to calling ABSInitializeEx with flags set to zero.	
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.2.2 ABSInitializeEx

```
ABS_STATUS ABSInitializeEx(
   IN ABS_DWORD dwFlags
)
```

Description	Initialize the BSAPI.DLL library. BSAPI.DLL must be initialized before you can call any other function. It is called typically during application startup. This function if more general version of ABSInitialize.	
Parameters		
dwFlags		On Windows, flags ABS_INIT_FLAG_NT_SERVICE, ABS_INIT_FLAG_FORCE_LOCAL_SENSOR and ABS_INIT_FLAG_FORCE_REMOTE_SENSOR are supported. On other systems, no flags are currently supported. See Flags for ABSInitializeEx (ABS_INIT_FLAG_xxxx) and Support for Terminal Services and Citrix for more information.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.2.3 ABSTerminate

```
ABS_STATUS ABSTerminate(
     void
)
```

Description	Uninitialize the BSAPI.DLL library. Must not be called while any connections to FM are still open. It is not obligatory to call this function, if the BSAPI.DLL library should be kept initialized until the program exits, but it is recommended practice to do so.	
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.2.4 ABSOpen

```
ABS_STATUS ABSOpen(
   IN const ABS_CHAR *pszDsn
   OUT ABS_CONNECTION *phConnection
)
```

Description	Open a new session with a FM.	
Parameters		
pszDsn		Zero-terminated ASCII string describing the FM connection parameters.
phConnection		Resulting connection handle. At the end of the connection it should be closed with .
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

Remarks	To close the connection you should call ABSClose.
	
	To open a connection through USB, no extra parameters are necessary.
	
	Example:
	`  DSN = "usb"`
	
	`"device=X"` (for USB only) This option can be used for opening specified device if more than one are simultaneously attached to the system. X is the device name string identifying unambiguously required device and it can be obtained from call of function ABSEnumerateDevices (actually ABSEnumerateDevices will give whole DSN string). Note that X is dependent on host current system configuration.
	
	Example:
	`DSN = "usb,device=\\?\usb#vid_0483&pid_2016#5&20890ddc&0&1#`
	`{d5620e51-8478-44bd-867e-aac02f883a00}"`	

2.2.5 ABSClose

```
ABS_STATUS ABSClose(
   IN ABS_CONNECTION hConnection
)
```

Description	Close a connection previously opened by ABSOpen.	
Parameters		
hConnection		Connection handle of the connection to be closed.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.
Remarks	Every successful call of ABSOpen should be paired with a call to ABSClose.	

2.2.6 ABSEnumerateDevices

```
ABS_STATUS ABSEnumerateDevices(
   IN const ABS_CHAR *pszEnumDsn
   OUT ABS_DEVICE_LIST **ppDeviceList
)
```

Description	Enumerate currently connected fingerprint devices.	
Parameters		
pszEnumDsn		Zero terminated ASCII string describing the connection interface, where the enumeration should be performed. For example to enumerate all devices connected to USB use string "usb".
ppDeviceList		Address of the pointer, which will be set to point to the list of found devices. The data has to be freed by a call to ABSFree.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.
Remarks	Note: Currently only devices on USB interface can be enumerated.	

2.2.7 ABSGetDeviceProperty

```
ABS_STATUS ABSGetDeviceProperty(
   IN ABS_CONNECTION hConnection
   IN ABS_DWORD dwPropertyId
   OUT ABS_DATA **ppPropertyData
)
```

Description	Return data describing some property of device associated with the current open session.	
Parameters		
hConnection		Handle to the connection to FM.
dwPropertyId		One of ABS_DEVPROP_xxxx constants, specifying what device property the caller is interested in.
ppPropertyData		Address of a pointer which will be set to point to a data block. The content of the data depends on dwPropertyId. The data has to be freed by a call to ABSFree. See documentation of the constants for specific information.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.2.8 ABSFree

```
void ABSFree(
   IN void *Memblock
)
```

Description	Use this function to releasing memory allocated by other BSAPI.DLL functions.
Parameters	
Memblock	Address of a memory block to be released. It has no effect if this parameter is set to NULL.

2.3 Biometric Functions

We will describe the biometric functions in this chapter.

2.3.1 ABSEnroll

```
ABS_STATUS ABSEnroll(
   IN ABS_CONNECTION hConnection
   IN ABS_OPERATION *pOperation
   OUT ABS_BIR **ppEnrolledTemplate
   IN ABS_DWORD dwFlags
)
```

Description	Scan the live finger, process it into a fingerprint template and return it to the caller.	
Parameters		
hConnection		Handle to the connection to FM.
pOperation		See description of ABS_OPERATION.
ppEnrolledTemplate		Address of the pointer, which will be set to point to the resulting template (BIR). The template has to be discarded by a call to ABSFree.
dwFlags		Reserved for future use. Set to zero.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.3.2 ABSVerify

```
ABS_STATUS ABSVerify(
   IN ABS_CONNECTION hConnection
   IN ABS_OPERATION *pOperation
   IN ABS_DWORD dwTemplateCount
   IN ABS_BIR **pTemplateArray
   OUT ABS_LONG *pResult
   IN ABS_DWORD dwFlags
)
```

Description	This function captures sample from the FM, processes it into template and compares it with templates, specified by the pTemplateArray parameter and finds out the first template which matches the swiped finger.	
Parameters		
hConnection		Handle to the connection to FM.
pOperation		See description of ABS_OPERATION.
dwTemplateCount		Count of templates in the pTemplateArray.
pTemplateArray		Pointer to the array of pointers to templates.
pResult		Pointer to memory location, where result of the comparing will be stored. The result is index into the pTemplateArray, determining the matching template, or -1 if no template matches.
dwFlags		Bitmask specifying flags, which modify slightly behavior of the function. This function supports flags ABS_FLAG_NOTIFICATION and ABS_FLAG_AUTOREPEAT. If ABS_FLAG_NOTIFICATION is set, the verification operation does not shows any GUI feedback until the finger is swiped. This can be useful for applications doing their own work; and only when user swipes, the application processes some special action. Until the user swipes, he is not disrupted with any dialog. Showing/hiding of the dialog is controlled by messages ABS_MSG_DLG_SHOW and ABS_MSG_DLG_HIDE. If flag ABS_FLAG_AUTOREPEAT is used, the verification is automatically restarted when the user's swipe does not match any template in pTemplateArray. This is better then calling the function in a loop until the user's swipe matches some template in the pTemplateArray, because this can prevent a GUI feedback dialog from hiding and showing between the calls.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.3.3 ABSVerifyMatch

```
ABS_STATUS ABSVerifyMatch(
   IN ABS_CONNECTION hConnection
   IN ABS_BIR *pEnrolledTemplate
   IN ABS_BIR *pVerificationTemplate
   OUT ABS_BOOL *pResult
   IN ABS_DWORD dwFlags
)
```

Description	Compares whether two given templates match or not.	
Parameters		
hConnection		Handle to the connection to FM.
pEnrolledTemplate		The fiest template to be compared. In the most common situation, when a template with enrollment purpose is being matched with a template with another purpose, the enrollment template has to be passed as this parameter.
pVerificationTemplate		The second template to be compared. In the most common situation, when a template with enrollment purpose is being matched with a template with another purpose, the latter template has to be passed as this parameter.
pResult		Output parameter to be set to result of the comparing. Set to ABS_TRUE if the two BIRs do match and ABS_FALSE if they do not.
dwFlags		Reserved for future use. Set to zero.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.3.4 ABSCapture

```
ABS_STATUS ABSCapture(
   IN ABS_CONNECTION hConnection
   IN ABS_OPERATION *pOperation
   IN ABS_DWORD dwPurpose
   OUT ABS_BIR **ppCapturedTemplate
   IN ABS_DWORD dwFlags
)
```

Description	This function captures sample for the purpose specified and creates a new fingerprint template from it.	
Parameters		
hConnection		Handle to the connection to FM.
pOperation		See description of ABS_OPERATION.
dwPurpose		A value indicate a purpose of the biometric data capture. It can be either ABS_PURPOSE_ENROLL or ABS_PURPOSE_VERIFY. ABS_PURPOSE_UNDEFINED is not allowed. Note that calling ABSCapture() with dwPurpose set to ABS_PURPOSE_ENROLL is obsolete and it is functionally equivalent to calling ABSEnroll().
ppCapturedTemplate		Pointer which is set to newly allocated template. Caller is responsible to release the memory with ABSFree.
dwFlags		Bitmask specifying flags, which modify slightly behavior of the function. This function supports only flag ABS_FLAG_NOTIFICATION. If it is set, the verification operation does not show any GUI feedback until the finger is swiped. This can be useful for applications doing their own work; and only when user swipes, the application processes some special action. Until the user swipes, he is not disrupted with any dialog. Showing/hiding of the dialog is controlled by messages ABS_MSG_DLG_SHOW and ABS_MSG_DLG_HIDE. The flag ABS_FLAG_NOTIFICATION can be used only in case that dwPurpose is set to ABS_PURPOSE_VERIFY.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.3.5 ABSCheckLatent

```
ABS_STATUS ABSCheckLatent(
   IN ABS_CONNECTION hConnection
   IN void *pReserved
   OUT ABS_BOOL *pboIsLatent
   IN ABS_DWORD dwFlags
)
```

Description	Perform antilatent check. Note that ABSEnroll and ABSVerify perform the check implicitely unless it is disabled with global parameter ABS_PARAM_LATENT_CHECK, so you should not call this function after ABSVerify() and ABSEnroll() after calling those functions (assuming the global parameter ABS_PARAM_LATENT_CHECK is turned on). See chapter Anti-latent Checking for more information.	
Parameters		
hConnection		Handle to the connection to FM.
pReserved		Reserved for future use. Set to NULL.
pboIsLatent		Pointer to ABS_BOOL, where the result of the check operation is stored. It is set to ABS_TRUE, if the latest scanned finger was detected as a latent finger; ABS_FALSE otherwise. Note that the value of the output variable is undefined if the check operation is not performed, i.e. when dwFlags == ABS_LATENT_OP_STORE.
dwFlags		Flags ABS_LATENT_OP_CHECK and ABS_LATENT_OP_STORE are supported. Note that using zero is quivalent to setting both flags (ABS_LATENT_OP_CHECK | ABS_LATENT_OP_STORE). When only flag ABS_LATENT_OP_CHECK is set, the function compares the most recent finger scan with (previously) stored scan and determines whether the most recent scan is latent or not. When only flag ABS_LATENT_OP_STORE is set, the most recent scan is stored so any future checks compare to this scan. When set to (ABS_LATENT_OP_CHECK | ABS_LATENT_OP_STORE) or to zero, then both check operation, and the store operation are performed (in this order).
Return value	ABS_STATUS	Result code.

2.3.6 ABSNavigate

```
ABS_STATUS ABSNavigate(
   IN ABS_CONNECTION hConnection
   IN ABS_OPERATION *pOperation
   IN ABS_DWORD dwFlags
)
```

Description	Switch FM to navigation mode (a.k.a. biometric mouse). Not all devices support this mode. If the navigation is not supported by the device, ABS_STATUS_NOT_SUPPORTED is returned. Please note there are few very old legacy devices where the navigation support is broken. With these devices, the function never sends ABS_MSG_NAVIGATE_CHANGE messages to the callback, making the navigation mode unusable. Remember that the function never returns ABS_STATUS_OK because it does not return until the function is canceled with ABSCancelOperation (or until an error occurs).	
Parameters		
hConnection		Handle to the connection to FM.
pOperation		See description of ABS_OPERATION.
dwFlags		Reserved for future use. Set to zero.
Return value	ABS_STATUS	Result code.

2.4 Image Grabbing Functions

There are 4 functions for retrieving fingerprint image from the sensor, each having its advantages and disadvantages. Here we summarize their differencies among the functions, in order to give you enough information to select the right one to fulfill your task.

Functions ABSGrab and ABSGrabImage are generally device independent, i.e. you don't need to have exact knowledge about what the device supports or not.

Functions ABSRawGrab and ABSRawGrabImage are device dependent and allow lower-level tuning of the image scanning process, including image quality checks and other attributes to be tuned.

Function ABSGrab offers only very limited possibility to choose desired image format, by using or not using a flag ABS_FLAG_HIGH_RESOLUTION.

Function ABSRawGrab allows to specify exact desired image format in a form of a record in grab profile. To use this, you have to know what image formats your device actually really supports. See documentation for ABS_PKEY_IMAGE_FORMAT constant and ABS_IFMT_xxxx constants for more details.

Functions ABSGrabImage and ABSRawGrabImage require that you describe the desired image format in a form of ABS_IMAGE_FORMAT structure. You can get list of supported formats with function ABSListImageFormats.

Functions ABSGrab and ABSGrabImage ask the user to swipe his finger until the resulted image passes some internal quality checks, i.e. there is some guaranty that you get image of real fingerprint.

In contrary, functions ABSRawGrab and ABSRawGrabImage return always after the first finger swipe. It's then callers responsibility to check image quality and (if he desires so) to call the function again.

2.4.1 ABSGrab

```
ABS_STATUS ABSGrab(
   IN ABS_CONNECTION hConnection
   IN ABS_OPERATION *pOperation
   IN ABS_DWORD dwPurpose
   OUT ABS_IMAGE **ppImage
   IN ABS_DWORD dwFlags
)
```

Description	Grabs image sample from the FM. Please note that unless an error occurs or it is canceled with ABSCancelOperation, the grab operation is automatically repeated until an image of some minimal quality is provided.
Parameters	
hConnection	Handle to the connection to FM.
pOperation	See description of ABS_OPERATION.
dwPurpose	A value indicate a purpose of the biometric data capture. It can be any ABS_PURPOSE_xxxx constant.
ppImage	Functions sets the pointer to newly allocated sample image. Use ABSFree to release the allocated memory.

dwFlags		Only flag ABS_FLAG_HIGH_RESOLUTION is supported.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.4.2 ABSRawGrab

```
ABS_STATUS ABSRawGrab(
   IN ABS_CONNECTION hConnection
   IN ABS_OPERATION *pOperation
   IN ABS_DWORD dwProfileSize
   IN ABS_PROFILE_DATA *pProfileData
   OUT ABS_IMAGE **ppImage
   OUT ABS_SWIPE_INFO **ppSwipeInfo
   IN ABS_DWORD dwFlags
)
```

Description	Grabs image sample from the FM. This function is similar to ABSGrab, but it is more low level. It allows to specify some special tuning parameters, thus tuning of the grab operation for specific purposes is possible. Those flags can specify what quality checks should be run, desired image format and other options. Please note that all these options are device specific. Various device models allow to tune various aspects of the grab operations, to various degree. Please prefer ABSGrab or ABSGrabImage whenever possible. Unlike ABSGrab this function waits for just one swipe and ends even if the resulting image has low quality. The caller can use the output parameters to further inspect quality of the resulted sample image.
Parameters	
hConnection	Handle to the connection to FM.
pOperation	See description of ABS_OPERATION.
dwProfileSize	Determines how many properties are in pProfileData.
pProfileData	Pointer to first member of profile data array. See description of ABS_PROFILE_DATA type for more information about the profile.
ppImage	Functions sets the pointer to newly allocated sample image. Use ABSFree to release the allocated memory.
ppSwipeInfo	If used (i.e. not set to NULL), an additional information about the swipe are provided to the caller. Use ABSFree to release the returned memory block. See description of ABS_SWIPE_INFO for more information.
dwFlags	Bitmask specifying flags, which modify slightly behavior of the function. Only flag ABS_FLAG_STRICT_PROFILE is supported. By default the profile data are respected to the degree possible i.e. those which are not supported by the device are silently ignored. In contrast, if flag ABS_FLAG_STRICT_PROFILE is set, the profile data is interpreted in a more strict way and the function returns

		ABS_STATUS_NOT_SUPPORTED if any specified requested tuning parameter or its particular value is not supported. Please note that requested image format (ABS_PKEY_IMAGE_FORMAT) is always interpreted in the strict way, i.e. the caller has to know which image formats are supported by the FM he uses.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.4.3 ABSListImageFormats

```
ABS_STATUS ABSListImageFormats(
   IN ABS_CONNECTION hConnection
   OUT ABS_DWORD *pdwCount
   OUT ABS_IMAGE_FORMAT **ppImageFormatList
   IN ABS_DWORD dwFlags
)
```

Description	Retrieves list of image formats supported by the FM. Functions ABSGrabImage and ABSRawGrabImage takes image format in the form of ABS_IMAGE_FORMAT as their parameter.	
Parameters		
hConnection		Handle to the connection to FM.
pdwCount		Count of image formats returned.
ppImageFormatList		Newly allocated array of image format structures is stored into this pointer. Use ABSFree to release the memory.
dwFlags		Reserved for future use. Set to zero.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.4.4 ABSGrabImage

```
ABS_STATUS ABSGrabImage(
   IN ABS_CONNECTION hConnection
   IN ABS_OPERATION *pOperation
   IN ABS_DWORD dwPurpose
   IN ABS_IMAGE_FORMAT *pImageFormat
   OUT ABS_IMAGE **ppImage
   OUT ABS_SWIPE_INFO **ppSwipeInfo
   IN void *pReserved
   IN ABS_DWORD dwFlags
)
```

Description	Grabs image sample from the FM. Please note that unless an error occurs or it is canceled with ABSCancelOperation, the grab operation is automatically repeated until an image of some minimal quality is provided.	
Parameters		
hConnection		Handle to the connection to FM.
pOperation		See description of ABS_OPERATION.
dwPurpose		A value indicate a purpose of the biometric data capture. It can be any ABS_PURPOSE_xxxx constant.
pImageFormat		Pointer to structure describing desired image format.
ppImage		Functions sets the pointer to newly allocated sample image. Use ABSFree to release the allocated memory.
ppSwipeInfo		If used (i.e. not set to NULL), an additional information about the swipeare provided to the caller. Use ABSFree to release the returned memory block. See description of ABS_SWIPE_INFO for more information.
pReserved		Reserved for future use. Set to NULL.
dwFlags		Reserved for future use. Set to zero.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.4.5 ABSRawGrabImage

```
ABS_STATUS ABSRawGrabImage(
   IN ABS_CONNECTION hConnection
   IN ABS_OPERATION *pOperation
   IN ABS_DWORD dwProfileSize
   IN ABS_PROFILE_DATA *pProfileData
   IN ABS_IMAGE_FORMAT *pImageFormat
   OUT ABS_IMAGE **ppImage
   OUT ABS_SWIPE_INFO **ppSwipeInfo
   IN void *pReserved
   IN ABS_DWORD dwFlags
)
```

Description	Grabs image sample from the FM. This function is similar to ABSGrabImage, but it is more low level. It allows to specify some special tuning parameters, thus tuning of the grab operation for specific purposes is possible. Those flags can specify what quality checks should be run and other options. Please note that all these options are device specific. Various device models allow to tune various aspects of the grab operations, to various degree. Prefer ABSGrab or ABSGrabImage whenever possible. Unlike ABSGrabImage this function waits for just one swipe and ends even if the resulting image has low quality. The caller can use the output parameters to further inspect quality of the resulted sample image. Note that if the grab profile uses key ABS_PKEY_IMAGE_FORMAT, it is ignored, as this functions always uses the format as specified by pImageFormat parameter.
Parameters	
hConnection	Handle to the connection to FM.
pOperation	See description of ABS_OPERATION.
dwProfileSize	Determines how many properties are in pProfileData.
pProfileData	Pointer to first member of profile data array. See description of ABS_PROFILE_DATA type for more information about the profile.
pImageFormat	Poitner to structure describing desired image format.
ppImage	Functions sets the pointer to newly allocated sample image. Use ABSFree to release the allocated memory.
ppSwipeInfo	If used (i.e. not set to NULL), an additional information about the swipeare provided to the caller. Use ABSFree to release the returned memory block. See description of ABS_SWIPE_INFO for more information.

pReserved		Reserved for future use. Set to NULL.
dwFlags		Bitmask specifying flags, which modify slightly behavior of the function. Only flag ABS_FLAG_STRICT_PROFILE is supported. By default the profile data are respected to the degree possible i.e. those which are not supported by the device are silently ignored. In contrast, if flag ABS_FLAG_STRICT_PROFILE is set, the profile data is interpreted in a more strict way and the function returns ABS_STATUS_NOT_SUPPORTED if any specified requested tuning parameter or its particular value is not supported. Please note that requested image format (ABS_PKEY_IMAGE_FORMAT) is always interpreted in the strict way, i.e. the caller has to know which image formats are supported by the FM he uses.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.5 Miscellaneous Functions

2.5.1 ABSCancelOperation

```
ABS_STATUS ABSCancelOperation(
   IN ABS_CONNECTION hConnection
   IN ABS_DWORD dwOperationID
)
```

Description	Cancels a running interactive operation. Function of the canceled operation returns ABS_STATUS_CANCELED.	
Parameters		
hConnection		Handle to the connection to FM.
dwOperationID		ID of the operation to be canceled, or zero to cancel the currently processed operation in the current thread. I.e. zero can be used only from callback to cancel the particular operation which called the callback. It is the only way how to cancel interactive operations which have OperationID set to zero. See description of member OperationID of structure ABS_OPERATION for more information.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.5.2 ABSSetAppData

```
ABS_STATUS ABSSetAppData(
   IN ABS_CONNECTION hConnection
   IN ABS_DATA *pAppData
)
```

Description	Stores arbitrary data on the FM. The data can be later retrieved with function ABSGetAppData. The data survive across BSAPI sessions, until the data are overwritten by next call to this function. Note that maximal length of the data is limited. The limit is device model dependent.	
Parameters		
hConnection		Handle to the connection to FM.
pAppData		The data to be stored on the device.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.5.3 ABSGetAppData

```
ABS_STATUS ABSGetAppData(
   IN ABS_CONNECTION hConnection
   OUT ABS_DATA **ppAppData
)
```

Description	Retrieves the data stored on the FM. See also description of function SetAppData.	
Parameters		
hConnection		Handle to the connection to FM.
ppAppData		Output parameter, to be set to the newly allocated structure ABS_DATA. Use ABSFree to release the allocated memory.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.5.4 ABSSetSessionParameter

```
ABS_STATUS ABSSetSessionParameter(
   IN ABS_CONNECTION hConnection
   IN ABS_DWORD dwParamID
   IN ABS_DATA *pParamValue
)
```

Description	Sets value of session-wide parameter. These settings influence behavior of certain BSAPI functions called in context of the current session. Please note that in the current version all parameters are global and can be set only with ABSSetGlobalParameter(). Therefore this functions returns ABS_STATUS_INVALID_PARAMETER as it does not support any dwParamID value currently.	
Parameters		
hConnection		Handle to the connection to FM.
dwParamID		ID of the parameter to set. See description of ABS_PARAM_xxxx constants.
pParamValue		Parameter value. Format and meaning of the data is parameter dependent. See description of particular ABS_PARAM_xxxx constant, you use as dwParamID for more information.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.5.5 ABSGetSessionParameter

```
ABS_STATUS ABSGetSessionParameter(
   IN ABS_CONNECTION hConnection
   IN ABS_DWORD dwParamID
   OUT ABS_DATA **ppParamValue
)
```

Description	Retrieves value of session-wide parameter. Please note that in the current version all parameters are global and can be set only with ABSSetGlobalParameter(). Therefore this functions returns ABS_STATUS_INVALID_PARAMETER as it does not support any dwParamID value currently.	
Parameters		
hConnection		Handle to the connection to FM.
dwParamID		ID of the parameter to retrieve. See description of ABS_PARAM_xxxx constants.
ppParamValue		Output parameter for the retrieved value. The function sets it to point to newly allocated ABS_DATA. Use ABSFree to release the memory. See description of ABS_PARAM_xxxx constants for meaning of particular values.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.5.6 ABSSetGlobalParameter

```
ABS_STATUS ABSSetGlobalParameter(
   IN ABS_DWORD dwParamID
   IN ABS_DATA *pParamValue
)
```

Description	Sets value of global-wide parameter. These settings influence behavior of certain BSAPI functions. Unlike ABSSetSessionParameter, the settings apply to all BSAPI functions called in a context of the process, despite the current session.	
Parameters		
dwParamID		ID of the parameter to set. See description of ABS_PARAM_xxxx constants.
pParamValue		Parameter value. Format and meaning of the data is parameter dependent. See description of particular ABS_PARAM_xxxx constant, you use as dwParamID for more information.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.5.7 ABSGetGlobalParameter

```
ABS_STATUS ABSGetGlobalParameter(
   IN ABS_DWORD dwParamID
   OUT ABS_DATA **ppParamValue
)
```

Description	Retrieves value of global-wide parameter.	
Parameters		
dwParamID		ID of the parameter to retrieve. See description of ABS_PARAM_xxxx constants.
ppParamValue		Output parameter for the retrieved value. The function sets it to point to newly allocated ABS_DATA. Use ABSFree to release the memory. See description of ABS_PARAM_xxxx constants for meaning of particular values.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.5.8 ABSSetLED

```
ABS_STATUS ABSSetLED(
   IN ABS_CONNECTION hConnection
   IN ABS_DWORD dwMode
   IN ABS_DWORD dwLED1
   IN ABS_DWORD dwLED2
)
```

Description	This function allows the application to control the state and behavior of two user interface LEDs, which can be optionally connected to the FM.	
Parameters		
hConnection		Handle to the connection to FM.
dwMode		Mode of the LEDs. Different modes define different behavior of the LEDs during specific operations, especially the biometrics. See the remarks below for more details.
dwLED1		Parameter defining the detailed behavior of the 1st LED. See the remarks below for more details.
dwLED2		Parameter defining the detailed behavior of the 2nd LED. See the remarks below for more details.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.
Remarks	The parameter dwMode can be set to one of the following constants. The meaning of the parameters dwLED1 and dwLED2 then depends on the chosen mode. **Mode ABS_LED_MODE_MANUAL:** This mode controls two LEDs of the device (if the device has them) in the manual mode. The two LEDs blink as specified by the parameters dwLED1 (controsl 1st LED) and dwLED2. Exact meaning of these parameters is described in the chapter Blinking with LEDs **Mode ABS_LED_MODE_MANUAL2:** This mode cannot be set by the function ABSSetLED. You have to use ABSSetLedEx(). **Mode ABS_LED_MODE_AUTO:** In this mode all LEDs are controled automatically by BSAPI. Set both dwLED1 and dwLED2 to zero. **Mode ABS_LED_MODE_OFF:** This mode causes all LEDs to be permanently turned OFF.	

	Set both dwLED1 and dwLED2 to zero.

2.5.9 ABSGetLED

```
ABS_STATUS ABSGetLED(
   IN ABS_CONNECTION hConnection
   OUT ABS_DWORD *dwMode
   OUT ABS_DWORD *pdwLED1
   OUT ABS_DWORD *pdwLED2
)
```

Description	This function allows the application to query the state and behavior of the two user interface LEDs, which can be optionally connected to the FM. For more information about the topic, see also documentation of function ABSSetLED.	
Parameters		
hConnection		Handle to the connection to FM.
dwMode		Returns a mode of the LEDs. See the remarks below for more details.
pdwLED1		Returns a value defining the detailed behavior of the 1st LED. See the remarks below for more details.
pdwLED2		Returns a value defining the detailed behavior of the 2nd LED. See the remarks below for more details.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.
Remarks	ABSGetLED returns the LED status as set by the latest call to ABSSetLED or ABSSetLedEx. If pdwLED1/pdwLED2 are not set to NULL, then depending on the current mode, the double-words pointed by them are set top the values used when the LEDs were set. Note that for LED modes which cannot be set by ABSSetLED (i.e. which can only by set by ABSSetLEdEx, the value of pointed doublewords is undefined. You have to use ABSGe-tLedEx() to retrieive mroe information about current settings of LEDs in that case.	

2.5.10 ABSSetLedEx

```
ABS_STATUS ABSSetLedEx(
   IN ABS_CONNECTION hConnection
   IN ABS_DWORD dwMode
   IN ABS_DATA *pLedParams
)
```

Description		
Parameters		
hConnection		Handle to the connection to FM.
dwMode		Mode of the LEDs. Different modes define different behavior of the LEDs during specific operations, especially the biometrics. See the remarks below for more details.
pLedParams		Data defining the detailed behavior of the LEDs, depending on the mode. See the remarks below for more details.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.
Remarks	The parameter dwMode can be set to one of the following constants. Interpretation of the parameter pLedParams then depends on the chosen mode. **Mode ABS_LED_MODE_MANUAL:** This mode controls two LEDs of the device (if the device has them) in the manual mode. The parameter pLedParams must point to ABS_DATA containing structure ABS_LED_PARAMS_MANUAL. **Mode ABS_LED_MODE_MANUAL2:** This mode controls two LEDs of the device (if the device has them) in the manual mode. The parameter pLedParams must point to ABS_DATA containing structure ABS_LED_PARAMS_MANUAL2. **Mode ABS_LED_MODE_AUTO:** In this mode all LEDs are controled automatically by BSAPI. The pointer pLedParams must be set to NULL or point to empty ABS_DATA. **Mode ABS_LED_MODE_OFF:** This mode causes all LEDs to be permanently turned OFF. The pointer pLedParams must be set to NULL or point to empty ABS_DATA.	

2.5.11 ABSGetLedEx

```
ABS_STATUS ABSGetLedEx(
   IN ABS_CONNECTION hConnection
   OUT ABS_DWORD *pdwMode
   OUT ABS_DATA **ppLedParams
)
```

Description		
Parameters		
hConnection		Handle to the connection to FM.
pdwMode		Returns a mode of the LEDs. See description of ABSSetLedEx() for more detailed description.
ppLedParams		Output parameter, to be set to the newly allocated structure of ABS_DATA. Use ABSFree to release the allocated memory.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.5.12 ABSBinarizeSampleImage

```
ABS_STATUS ABSBinarizeSampleImage(
   INOUT ABS_IMAGE *pGrayScaleImage
   OUT ABS_IMAGE **ppBinarizedImage
)
```

Description	The function converts gray-scale image (as obtain from callback, ABSGrab or ABSRawGrab) to binarized form, with only two colors. I.e. the binarized image sample has ColorCount set to 2. The two colors are then interpreted as black (1) and white (0). The binarized image is more suitable for displaying to the end-user because it usually looks better. Note that the conversion can be taken in-place or to newly allocated image structure depening if you set the parameter ppBinarizedImage to NULL or not.	
Parameters		
pGrayScaleImage		Pointer to the input, gray-scale image structure. Please note that this function does not support all image formats. Only formats having 8 bits per pixel (ABS_IMAGE::ColorCount == 256) and having resolution 381x381 DPI or 508x508 DPI. If the ppBinarizedImage is NULL, then the content of this structure is modified in-place.
ppBinarizedImage		Optional output parameter for retrieving the new, binarized sample image. If non-NULL, the converted image sample will be placed in newly allocated buffer pointed by this output parameter. Caller is then responsible for releasing the memory with ABSFree. If NULL, the original image pGrayScaleImage will be converted in-place.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.5.13 ABSGetLastErrorInfo

```
void ABSGetLastErrorInfo(
   OUT ABS_DWORD *pErrorCode
   OUT const ABS_CHAR **ppErrorMessage
)
```

Description	Retrieves additional information about last BSAPI error, which occurred in the current thread. Please note that information provided by this function is **not** intended to be displayed to the end user. The error messages are in English (they are never localized) and they are meant as a hint for developers to make problem diagnosis easier.
Parameters	
pErrorCode	Output parameter set to additional system dependent error code. Depending on system it might be errno or value returned by Get-LastError on Windows platforms or error code from any lower level library used by BSAPI.DLL. It might give a hint what's going wrong when diagnosing the problem.
ppErrorMessage	On output this is set to point to a buffer containing zero-terminated string with textual message. If no message is provided, it points to empty string so the caller does not need check it for NULL. The buffer is managed by BSAPI; do **not** use ABSFree to release it. Note that the buffer is valid only until other BSAPI call is performed in the same thread. After the next call, the buffer may be released or reused by BSAPI. If you need to remember the message, you have to copy it into your own buffer.

2.5.14 ABSEscape

```
ABS_STATUS ABSEscape(
   IN ABS_DWORD dwOpcode
   IN ABS_DATA *pInData
   OUT ABS_DATA **ppOutData
)
```

Description	Requests special function to be processed. Note that opcodes intended for use with ABSSession can not be used with ABSSessionEscape and vice versa.	
Parameters		
dwOpcode		Code of operation to perform. Currently only some codes are supported for use by UPEK partners. They are not intended for public audience, hence they are not described here.
pInData		Input data, passed to the function requested by dwOpcode. Format of the data depends on dwOpcode. Can be NULL if the dwOpcode does not require any input data.
ppOutData		Data passed back to the caller, as a result of the operation. Can be set to NULL if no data are passed back.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

2.5.15 ABSSessionEscape

```
ABS_STATUS ABSSessionEscape(
   IN ABS_CONNECTION hConnection
   IN ABS_DWORD dwOpcode
   IN ABS_DATA *pInData
   OUT ABS_DATA **ppOutData
)
```

Description	Requests special function to be processed. This function differs from ABSEscape in the fact that opcodes supported by this function only work in a context of single BSAPI session. Note that opcodes intended for use with ABSSessionEscape can not be used with ABSEscape and vice versa.	
Parameters		
hConnection		Handle to the connection to FM.
dwOpcode		Code of operation to perform. Currently only some codes are supported for use by UPEK partners. They are not intended for public audience, hence they are not described here.
pInData		Input data, passed to the function requested by dwOpcode. Format of the data depends on dwOpcode. Can be NULL if the dwOpcode does not require any input data.
ppOutData		Data passed back to the caller, as a result of the operation. Can be set to NULL if no data are passed back.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

3 BSGUI.DLL Functions

3.1 Using BSGUI.DLL

BSGUI.DLL provides a default ABS_CALLBACK implementation for BSAPI.DLL.

To use the default callback implementation, link your application with both BSAPI.DLL and BSGUI.DLL. Then, whenever you start any interactive operation, set member Callback of structure ABS_OPERATION to pointer to function ABSDefaultCallback from BSGUI.DLL.

When installing the application, file BSGUI.ZIP must be placed to the same directory as the BSGUI.DLL.

Please note that BSGUI.DLL (with BSGUI.ZIP) is available only for Windows platform.

3.2 GUI Customization

The BSGUI.DLL library loads graphics from file BSGUI.ZIP.

If you want to customize look and feel of the dialogs provided by BSGUI.DLL, replace some or all images in the BSGUI.ZIP package and, if necessary, change also layout in BIO.XML inside the BSGUI.ZIP.

3.3 Default Callback Implementation

3.3.1 ABSDefaultCallback

```
void ABSDefaultCallback(
   IN const ABS_OPERATION *pOperation
   IN ABS_DWORD dwMsgID
   IN void *pMsgData
)
```

Description	Default BSAPI callback implementation. It provides default implementation of callback, which can be passed into BSAPI interactive functions via ABS_OPERATION structure. Using this callback instead of your own implementation provides consistent look and feel across applications. You should never call this function directly. It's intended only to pass pointer to the function into the BSAPI functions as member Callback of structure ABS_OPERATION. For more information, see the documentation of type ABS_CALLBACK and structure ABS_OPERATION.
Parameters	
pOperation	Pointer to ABS_OPERATION structure used when calling the interactive biometric operation. The caller of the interactive operation can use member Context of the structure to pass data into the default callback. ABSDefaultCallback expects the data in the form of ABS_DEFAULT_CALLBACK_CONTEXT structure.

	The Context pointer can be set to NULL. In that case, default behavior is used.
dwMsgID	ID of message. See description of ABS_MSG_xxxx constants.
pMsgData	Pointer to data with additional information related with the message. Its meaning is message dependent. Refer to documentation of specific ABS_MSG_xxxx constants.

3.4 ABS_DEFAULT_CALLBACK_CONTEXT

Structure to be optionally passed as a context data into ABSDefaultCallback.

This allows to tune exact behavior of the default callback implementation. To use it, setup the structure members and set Context of ABS_OPERATION to address of the structure.

The caller of the biometric operation must guarantee that the pointer to the structure, passed in through ABS_OPERATION structure, remains valid until the biometric operation is over.

```
typedef struct abs_default_callback_context {
    ABS_DWORD Version;
    HWND ParentWindow;
    ABS_DWORD Flags;
} ABS_DEFAULT_CALLBACK_CONTEXT
```

Description	
Version	Version of the structure. Set to 1.
ParentWindow	Set to handle of parent window or NULL. When set to NULL, actually active window is used as the parent window.
Flags	Bitmask of flags. Currently only flag ABS_DEFAULT_CALLBACK_FLAG_ENABLE_SOUND is supported.

3.5 Flags for ABS_DEFAULT_CALLBACK_CONTEXT (ABS_DEFAULT_CALLBACK_FLAG_xxxx)

The following flag can be used in structure ABS_DEFAULT_CALLBACK_CONTEXT.

ABS_DEFAULT_CALLBACK_FLAG_ENABLE_SOUND	0x1
Enables the callback to play a sound on success.	

4 BSSRV.DLL Functions

4.1 General Description

BSSRV.DLL is library, which provides access to a biometric operations without a direct need of communication with fingerprint sensor device. Typical use case is server side of an application of server-client architecture where clients ask users swipe his finger on fingerprint sensor, but the biometry is performed on the server (where, for example, a database of fingerprint templates might be managed centraly).

The library uses conventions very similar conventions like BSAPI.DLL. Function declarations are declared in header bssrv.h. Various types, constants and error codes are shared with BSAPI.DLL.

Note however that the BSSRV.DLL and the header bssrv.h are only available within full BSAPI SDK. Lite version of the SDK does not contain these files.

4.1.1 Error Handling

Almost all BSSRV.DLL functions return a status code **ABS_STATUS**. Code **ABS_STATUS_OK** (zero) means success. All other values denote an error condition.

You may call ABSSrvGetLastErrorInfo to retrieve more information about the error condition. Note that the information is intended for application and library developers and it's not intended to be presented to end users.

If any BSSRV.DLL function fails, it frees any resources it might allocate. Values of output parameters are defined only if the function succeeds i.e. if it returns ABS_STATUS_OK.

4.1.2 Memory Management

Some BSSRV.DLL functions allocate memory returned via output parameter to the calling application. The application must use ABSSrvFree to free memory allocated by BSSRV.DLL in these cases.

4.1.3 Multithreading

In general, BSSRV.DLL is thread-safe. You can call BSSRV.DLL functions concurrently from multiple threads.

The only exception are functions ABSSrvInitialize and ABSSrvTerminate. These two functions are **not** thread-safe. This is usually not a problem, because they are usually called as part of application initialization and termination respectively.

4.1.4 BSAPI.DLL and BSSRV.DLL

In general, there should be no need to link one program to both BSAPI.DLL and BSSRV.DLL. If you need to link with BSAPI.DLL, then there is no need to use BSSRV.DLL in the same program probably.

One exception to the rule might be if you have one program which serves to both purposes in server-client architecture, i.e. when you have one program binary which is (depending on some configuration) either server or client.

However, if your application is linked with both, BSAPI.DLL and BSSRV.DLL, you cannot to free a memory allocated by one library with a function from the second. Accordingly you cannot resolve an error condition in one library with calls to second library.

For example. If you call a BSAPI.DLL function you cannot resolve its error by calling ABSSrvGetLastErrorInfo, but you have to use ABSGetLastErrorInfo.

4.2 Application General Functions

4.2.1 ABSSrvInitialize

```
ABS_STATUS ABSSrvInitialize(
     void
)
```

Description	Initialize the BSSRV library. Must be called before any other function, typically during of the application's startup.	
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

4.2.2 ABSSrvTerminate

```
ABS_STATUS ABSSrvTerminate(
     void
)
```

Description	Uninitialize the BSSRV library.	
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

4.2.3 ABSSrvGetLastErrorInfo

```
void ABSSrvGetLastErrorInfo(
   OUT ABS_DWORD *pErrorCode
   OUT const ABS_CHAR **ppErrorMessage
)
```

Description	Retrieves additional information about last BSAPI error, which occurred in the current thread. Please note that information provided by this function is **not** intended to be displayed to the end user. The error messages are in English (they are never localized) and they are meant as a hint for developers to make problem diagnosis easier.
Parameters	
pErrorCode	Output parameter set to additional system dependent error code. Depending on system it might be errno or value returned by Get-LastError on Windows platforms or any other error code. It might give developer a hint what's going wrong.
ppErrorMessage	On output this is set to point to a buffer containing zero-terminated string with textual message. If no message is provided, it points to empty string so the caller does not need check it for NULL. The buffer is managed by BSAPI; do **not** use ABSFree to release it. Note that the buffer is valid only until other BSAPI call is performed in the same thread. After the next call, the buffer may be released or reused by BSAPI. If you need to remember the message, you have to copy it into your own buffer.

4.2.4 ABSSrvFree

```
void ABSSrvFree(
   IN void *Memblock
)
```

Description	Use this function to releasing memory allocated by other BSSRV.DLL functions.
Parameters	
Memblock	Address of a memory block to be released. It has no effect if this parameter is set to NULL.

4.3 Server-side Functions

4.3.1 ABSSrvVerifyMatch

```
ABS_STATUS ABSSrvVerifyMatch(
   IN void *pReserved
   IN ABS_BIR *pEnrolledTemplate
   IN ABS_BIR *pVerificationTemplate
   OUT ABS_BOOL *pResult
   IN ABS_DWORD dwFlags
)
```

Description	Compares whether two given templates match or not.	
Parameters		
pReserved		Reserved for future use. Set to NULL.
pEnrolledTemplate		The first template to be compared. In the most common situation, when a template with enrollment purpose is being matched with a template with another purpose, the enrollment template has to be passed as this parameter.
pVerificationTemplate		The second template to be compared. In the most common situation, when a template with enrollment purpose is being matched with a template with another purpose, the latter template has to be passed as this parameter.
pResult		Output parameter to be set to result of the comparing. Set to ABS_TRUE if the two BIRs do match and ABS_FALSE if they do not.
dwFlags		Reserved for future use. Set to zero.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

4.4 Miscellaneous Functions

4.4.1 ABSSrvSetGlobalParameter

```
ABS_STATUS ABSSrvSetGlobalParameter(
   IN ABS_DWORD dwParamID
   IN ABS_DATA *pParamValue
)
```

Description	Sets value of global-wide parameter. These settings influence behavior of certain BSSRV functions.	
Parameters		
dwParamID		ID of the parameter to set. See description of ABS_PARAM_xxxx constants. Please note that only ABS_PARAM_MATCH_LEVEL parameter is supported.
pParamValue		Parameter value. Format and meaning of the data is parameter dependent. See description of particular ABS_PARAM_xxxx constant, you use as dwParamID for more information.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

4.4.2 ABSSrvGetGlobalParameter

```
ABS_STATUS ABSSrvGetGlobalParameter(
   IN ABS_DWORD dwParamID
   OUT ABS_DATA **ppParamValue
)
```

Description	Retrieves value of global-wide parameter.	
Parameters		
dwParamID		ID of the parameter to retrieve. See description of ABS_PARAM_xxxx constants. Please note that only ABS_PARAM_MATCH_LEVEL parameter is supported.
ppParamValue		Output parameter for the retrieved value. The function sets it to point to newly allocated ABS_DATA. Use ABSSrvFree to release the memory. See description of ABS_PARAM_xxxx constants for meaning of particular values.
Return value	ABS_STATUS	Result code. ABS_STATUS_OK (0) means success.

5 Declarations

5.1 Basic Types

typedef Signed integer type (1 byte)	char	ABS_CHAR
typedef Unsigned integer type (1 byte)	unsigned char	ABS_BYTE
typedef Signed integer type (2 bytes)	short	ABS_SHORT
typedef Unsigned integer type (2 bytes)	unsigned short	ABS_WORD
typedef Signed integer type (4 bytes)	int	ABS_LONG
typedef Unsigned integer type (4 bytes)	unsigned int	ABS_DWORD
typedef Boolean value (zero, non-zero)	int	ABS_BOOL
typedef Return status	ABS_LONG	ABS_STATUS
typedef Connection handle. It represents a session with FM.	ABS_DWORD	ABS_CONNECTION

5.2 Specific Types

5.2.1 ABS_DATA

The ABS_DATA structure is used to associate any arbitrary long data block with the length information.

```
typedef struct abs_data {
    ABS_DWORD Length;
    ABS_BYTE Data[ABS_VARLEN];
} ABS_DATA
```

Description	
Length	Length of the Data field in bytes.
Data[ABS_VARLEN]	The data itself, variable length.

5.2.2 ABS_BIR_HEADER

The header of the BIR. This type is equivalent to BioAPI's structure BioAPI_BIR_HEADER.

In the typical use the BIR is handled as an opaque data, it is not necessary to know the structure of its header. However, we document it here for completeness. The values provided below are the standard values used in BIRs produced by the FM.

Please refer to BioAPI documentation for details.

Note that all members of the ABS_BIR_HEADER are always in little endian byte order. This has two important impacts:

- The template has exactly same binary representation, when stored to some storage or database, so they may be used on all platforms despite byte order the platform uses.
- When using values of the structure, you must convert the values to the natural byte order of the platform you use.

```
typedef struct abs_bir_header {
    ABS_DWORD Length;
    ABS_BYTE HeaderVersion;
    ABS_BYTE Type;
    ABS_WORD FormatOwner;
    ABS_WORD FormatID;
    ABS_CHAR Quality;
    ABS_BYTE Purpose;
    ABS_DWORD FactorsMask;
} ABS_BIR_HEADER
```

Description	
Length	Length of Header + Opaque Data
HeaderVersion	HeaderVersion = 1
Type	Type = 4 (BioAPI_BIR_DATA_TYPE_PROCESSED)
FormatOwner	FormatOwner = 0x12 (STMicroelectronics)
FormatID	FormatID = 0
Quality	Quality = -2 (BioAPI_QUALITY is not supported)
Purpose	Purpose (BioAPI_PURPOSE_xxxx, ABS_PURPOSE_xxxx). The corresponding BioAPI and BSAPI constants have the same values.
FactorsMask	FactorsMask = 0x08 (BioAPI_FACTOR_FINGERPRINT)

5.2.3 ABS_BIR

A container for biometric data.

The abbreviation BIR stands for Biometric Identification Record. In BSAPI it represents a fingerprint template, but potentially can contain other data as well, e.g. audit data. BIR consists of a header, followed by the opaque data and optionally by a signature. This type is binary compatible with BioAPI's BioAPI_BIR. The only difference is, that in BioAPI_BIR the data is divided into four separate memory blocks, while ABS_BIR keeps all the data together.

```
typedef struct abs_bir {
    ABS_BIR_HEADER Header;
    ABS_BYTE Data[ABS_VARLEN];
} ABS_BIR
```

Description	
Header	BIR header
Data[ABS_VARLEN]	The data composing the fingerprint template.

5.2.4 ABS_OPERATION

Holds common data used by all interactive operation functions.

```
typedef struct abs_operation {
    ABS_DWORD OperationID;
    void* Context;
    ABS_CALLBACK Callback;
    ABS_LONG Timeout;
    ABS_DWORD Flags;
} ABS_OPERATION
```

Description	
OperationID	Unique operation ID or zero. When set to non-zero, the value identifies the operation. You can then use the ID to cancel the operation with ABSCancelOperation, even from other thread. Please note that its caller's responsibility to assign the IDs so that in the context of one session no concurrent interactive operation (i.e. in other thread) has the same value. Otherwise the operation fails immidiatly with ABS_STATUS_INVALID_PARAMETER. If set to zero, you can cancel the operation only from its callback (passing a zero as the parameter for ABSCancelOperation).
Context	User defined pointer, passed into the operation callback. BSAPI does not interpret nor dereferences the pointed data in any way.
Callback	Pointer to application-defined function, implementing operation callback. This allows application developers to provide user interface which informs user how the operation processes and prompts him to do something, e.g. put his finger on the FM sensor. See documentation of ABS_CALLBACK for more detailed information.
Timeout	Timeout of user's inactivity in milliseconds If the interactive operation expects some user's activity and it's not detected for the time specified, the operation is interrupted and the operation function returns ABS_STATUS_TIMEOUT. Note that operation being in suspended state (i.e. when some other concurrent operation acquired the sensor) by definition does not expect user's activity on the sensor. Hence the time counting for suspended operation is held up. When the user's activity on sensor is detected, the time counter is reset. I.e. to experience the timeout, the user must not touch the sensor for the period of time as set by the Timeout value.

	Value 0 denotes no timeout (user's inactivity never causes the operation to be interrupted). Value -1 denotes to use the default timeout (device dependent).
Flags	Bitmask of flags, tuning the operation process. See description of constants ABS_OPERATION_FLAG_xxxx for more information.

5.2.5 ABS_PROFILE_DATA

Profile data for tuning raw grab operation (ABSRawGrab).

This allows to set various special modes and attributes of the raw grab, including various FM dependent ones.

Function ABSRawGrab takes pointer to array of structure ABS_PROFILE_DATA. (Other parameter specifies number of items in the profile array.) Each record in the array is composed of key-value pair.

The key specifies what attribute/parameter of the raw grab operation to tune and the value specifies how to tune that attribute/parameter. Meaning of the value and range of accepted values depends on the particular key and capabilities of the FM.

```
typedef struct abs_profile_data {
    ABS_DWORD Key;
    ABS_DWORD Value;
} ABS_PROFILE_DATA
```

Description	
Key	Profile key.It can be any constant ABS_PKEY_xxxx
Value	Value, key dependent.

5.2.6 ABS_SWIPE_INFO

This structure provides various informations about the swipe, from ABSRawGrab, ABSGrabImage or ABSRawGrabImage functions.

Please note that in future version of BSAPI the ABSRawGrab can return the information about the swipe in other format, then defined by this structure. See description of ABSRawGrab and its parameter ppSwipeInfo for more information.

```
typedef struct abs_swipe_info {
    ABS_DWORD Version;
    ABS_WORD Height;
    ABS_BYTE ReconstructionScore;
    ABS_BYTE ImageScore;
    ABS_DWORD MsgID;
    ABS_DWORD Flags;
    ABS_DWORD BackgroundColor;
} ABS_SWIPE_INFO
```

Description	
Version	Version of the structure. Current version is 1. I.e. if the first four bytes of the returned data is not 1, you cannot interpret the rest of the other data as structure SWIPE_INFO.
Height	Height of the fingerprint image in pixels.
ReconstructionScore	Reconstruction quality score, in range 0 - 100. The higher the value, the higher the quality of the image reconstruction.
ImageScore	Image quality score, in range 0 - 100. The higher the value, the higher the quality of the resulted image.
MsgID	Quality feedback message ID. Depending on the settings of the raw grab profile, various tests quality checks can be processed during the raw grab operation. If all the tests pass successfully (or if all of them are disabled) MsgId is set to ABS_MSG_PROCESS_SUCCESS. If any quality check failed, the value is set to the most important/relevant callback message ID (see constants ABS_MSG_QUALITY_xxxx).
Flags	Bitmask indicating various aspects of the swipe. See constants ABS_SWIPE_FLAG_xxxx.
BackgroundColor	Background color in the swiped sample image. Exact color depends on the sample image the ABS_SWIPE_INFO is related to. Value of 0 means black color, value (ABS_IMAGE::ColorCount - 1) means white color. Other grayscale colors are spread between black and white.

	If the background color could not be determined, BackgroundColor is set to 0xFFFFFFFF.

5.2.7 ABS_IMAGE_FORMAT

Type ABS_IMAGE_FORMAT desribes desired image format for functions ABSGrabImage and ABSRawGrabImage.

Use function ABSListImageFormats to retrieve list of available formats.

The resolution is always in DPI (dots per inch). Scan resolution is a resolution of the sensor during the scan. Image resolution is resolution of the resulted image. They are the same unless the sensor subsamples the scanned image or when the information about subsampling is not available for the given piece of hardware.

```
typedef struct abs_image_format {
    ABS_WORD ScanResolutionH;
    ABS_WORD ScanResolutionV;
    ABS_WORD ImageResolutionH;
    ABS_WORD ImageResolutionV;
    ABS_BYTE ScanBitsPerPixel;
    ABS_BYTE ImageBitsPerPixel;
} ABS_IMAGE_FORMAT
```

Description	
ScanResolutionH	Horizontal scan resolution, in dots per inch (DPI).
ScanResolutionV	Vertical scan resolution, in dots per inch (DPI).
ImageResolutionH	Horizonatal resolution of resulted image, in dots per inch (DPI).
ImageResolutionV	Vertical resolution of resulted image, in dots per inch (DPI).
ScanBitsPerPixel	Scan bits per pixel.
ImageBitsPerPixel	Bits per pixel of resulted image.If this value is N, then the resulted ABS_IMAGE::ColorCount is N-th power of two.

5.2.8 ABS_IMAGE

Type ABS_IMAGE holds data representing one sample image of swiped finger.

Functions ABSCapture, ABSGrab and ABSRawGrab use this structure. Also certain messages sent to ABS_CALLBACK can have sample image in form of this structure passed as additional data.

```
typedef struct abs_image {
    ABS_DWORD Width;
    ABS_DWORD Height;
    ABS_DWORD ColorCount;
    ABS_DWORD HorizontalDPI;
    ABS_DWORD VerticalDPI;
    ABS_BYTE ImageData[ABS_VARLEN];
} ABS_IMAGE
```

Description	
Width	Width of the image in pixels.
Height	Height of the image in pixels.
ColorCount	Maximal color count of the image.
HorizontalDPI	Horizontal resolution of the image (dots per inch).
VerticalDPI	Vertical resolution of the image (dots per inch).
ImageData[ABS_VARLEN]	Color values of all pixels. ImageData is an array of (Width * Height) bytes. Each pixel is represented by one byte. First (Width) bytes represent first row of pixels (from left to right ordering) and the subsequent row follows one by one, without any gaps. Value of each byte denotes a grayscale color. Colors are numbered, 0 meaning black and (colorCount - 1) meaning white. Other gray colors are linearly spread in the range between black and white.

5.2.9 ABS_LED_PARAMS_MANUAL

Parameters for LED blinking mode ABS_LED_MODE_MANUAL.

```
typedef struct abs_led_params_manual {
    ABS_DWORD Version;
    ABS_DWORD Led1;
    ABS_DWORD Led2;
    ABS_DWORD Flags;
} ABS_LED_PARAMS_MANUAL
```

Description	
Version	Version of this structure. Set to 1.
Led1	LED1 parameters.See the description in the chapter Blinking with LEDs for its meaning.
Led2	LED2 parameters.See the description in the chapter Blinking with LEDs for its meaning.
Flags	Reserved, set to zero.

5.2.10 ABS_LED_PARAMS_MANUAL2

Parameters for LED blinking mode ABS_LED_MODE_MANUAL2.

```
typedef struct abs_led_params_manual2 {
    ABS_DWORD Version;
    ABS_WORD Drv1_OnPeriod;
    ABS_WORD Drv1_OffPeriod;
    ABS_BYTE Drv1_BrCapHi;
    ABS_BYTE Drv1_BrCapLo;
    ABS_BYTE Drv1_BrStepX;
    ABS_BYTE Drv1_BrStepY;
    ABS_DWORD Drv1_Flags;
    ABS_WORD Drv2_OnPeriod;
    ABS_WORD Drv2_OffPeriod;
    ABS_BYTE Drv2_BrCapHi;
    ABS_BYTE Drv2_BrCapLo;
    ABS_BYTE Drv2_BrStepX;
    ABS_BYTE Drv2_BrStepY;
    ABS_DWORD Drv2_Flags;
    ABS_BYTE Drv3_TruthTable;
    ABS_BYTE Led1_Mapping;
    ABS_BYTE Led2_Mapping;
    ABS_BYTE Led3_Mapping;
    ABS_DWORD Flags;
} ABS_LED_PARAMS_MANUAL2
```

Description	
Version	Version of this structure (= 1).
Drv1_OnPeriod	Driver 1: Length of "ON" period in 4ms units (range 0-2047).
Drv1_OffPeriod	Driver 1: Length of "OFF" period in 4ms units (range 0-2047).
Drv1_BrCapHi	Driver 1: Maximum brightness value (range 0-63).
Drv1_BrCapLo	Driver 1: Minimum brightness value (range 0-63).
Drv1_BrStepX	Driver 1: Length of brightness change step in 4ms units (range 0-63, +1 added).
Drv1_BrStepY	Driver 1: Amount levels the LED brightness level will change at each 'BrStepX' (range 0-63).
Drv1_Flags	Driver 1: Additional flags, see ABS_LEDDRV_FLAG_xxx.
Drv2_OnPeriod	Driver 2: Length of "ON" period in 4ms units (range 0-2047).
Drv2_OffPeriod	Driver 2: Length of "OFF" period in 4ms units (range 0-2047).
Drv2_BrCapHi	Driver 2: Maximum brightness value (range 0-63).
Drv2_BrCapLo	Driver 2: Minimum brightness value (range 0-63).
Drv2_BrStepX	Driver 1: Length of brightness change step in 4ms units (range 0-63, +1 added).
Drv2_BrStepY	Driver 2: Amount levels the LED brightness level will change at each 'BrStepX' (range 0-63).
Drv2_Flags	Driver 2: Additional flags, see ABS_LEDDRV_FLAG_xxx.

Drv3_TruthTable	Definition of the Boolean combination of Driver 1 and Driver 2 that generates Driver 3. • Bit 0: Drv3 value when Drv1=0, Drv2=0 • Bit 1: Drv3 value when Drv1=1, Drv2=0 • Bit 2: Drv3 value when Drv1=0, Drv2=1 • Bit 3: Drv3 value when Drv1=1, Drv2=1
Led1_Mapping	LED1 mapping - 0=Drv1, 1=Drv3.
Led2_Mapping	LED2 mapping - 0=Drv2, 1=Drv3.
Led3_Mapping	LED3 mapping - 0=off, 1=Drv1, 2=Drv2, 3=Drv3.
Flags	Reserved, set to zero.

5.2.11 ABS_PROCESS_DATA

This structure is a container for additional data associated with ABS_MSG_PROCESS_xxxx messages sent to callback of an interactive operation.

Note that some message ABS_MSG_PROCESS_xxxx use more specific structure, however all are binary compatible with ABS_PROCESS_DATA i.e. pointer to them can be safely cast to pointer to ABS_PROCESS_DATA.

```
typedef struct abs_process_data {
    ABS_DWORD ProcessID;
} ABS_PROCESS_DATA
```

Description	
ProcessID	ID of process stage. See ABS_PROCESS_xxxx constants.

5.2.12 ABS_PROCESS_BEGIN_DATA

This structure is a container for additional data associated with ABS_MSG_PROCESS_BEGIN message sent to callback of an interactive operation.

```
typedef struct abs_process_begin_data {
    ABS_DWORD ProcessID;
    ABS_DWORD Step;
    ABS_DWORD StepCount;
} ABS_PROCESS_BEGIN_DATA
```

Description	
ProcessID	ID of process stage. See ABS_PROCESS_xxxx constants.
Step	Step number. Some operations are composed of multiple steps, e.g. consolidated enroll-ment where user has to swipe multiple times. First step is always marked with zero.
StepCount	Count of child steps of this process. If the count is not known (e.g. in the case of dynamic enrollment), then it is set to zero.

5.2.13 ABS_PROCESS_PROGRESS_DATA

This structure is a container for additional data associated with ABS_MSG_PROCESS_PROGRESS message sent to callback of an interactive operation.

```
typedef struct abs_process_progress_data {
    ABS_DWORD ProcessID;
    ABS_DWORD Percentage;
} ABS_PROCESS_PROGRESS_DATA
```

Description	
ProcessID	ID of process stage. See ABS_PROCESS_xxxx constants.
Percentage	Determines percentage of the process completeness. The value is in the range 0 - 100. If the percentage is not applicable to the process, it is set to 0xffffffff.

5.2.14 ABS_PROCESS_SUCCESS_DATA

This structure is a container for additional data associated with ABS_MSG_PROCESS_SUCCESS message sent to callback of an interactive operation.

```
typedef struct abs_process_success_data {
    ABS_DWORD ProcessID;
    ABS_IMAGE* SampleImage;
    ABS_BIR* Template;
} ABS_PROCESS_SUCCESS_DATA
```

Description	
ProcessID	ID of process stage. See ABS_PROCESS_xxxx constants.
SampleImage	Pointer to scanned image. Can be NULL if no image is associated with the message.
Template	Pointer to processed template. Can be NULL if no template is associated with the message.

5.2.15 ABS_NAVIGATION_DATA

This structure is a container for additional data associated with ABS_MSG_NAVIGATE_CHANGE message sent to callback of an interactive operation.

```
typedef struct abs_navigation_data {
    ABS_LONG DeltaX;
    ABS_LONG DeltaY;
    ABS_BOOL FingerPresent;
} ABS_NAVIGATION_DATA
```

Description	
DeltaX	Change of the virtual pointer's coordinates, in the horizontal direction.
DeltaY	Change of the virtual pointer's coordinates, in the vertical direction.
FingerPresent	ABS_TRUE if finger is present on the sensor, ABS_FALSE otherwise.

5.2.16 ABS_DEVICE_LIST_ITEM

Item of the device info list

```
typedef struct abs_device_list_item {
    ABS_CHAR DsnSubString[260];
    ABS_BYTE reserved[256];
} ABS_DEVICE_LIST_ITEM
```

Description	
DsnSubString[260]	String usable as part of DSN for ABSOpen to connect to this device.
reserved[256]	Reserved for future use.

5.2.17 ABS_DEVICE_LIST

The format of the data returned by ABSEnumerateDevices, it contains info about all enumerated devices. Please note, that the real output parameter from ABSEnumerateDevices has variable length – array List[] has NumDevices items.

```
typedef struct abs_device_list {
    ABS_DWORD NumDevices;
    ABS_DEVICE_LIST_ITEM List[ABS_VARLEN];
} ABS_DEVICE_LIST
```

Description	
NumDevices	Number of devices in the list
List[ABS_VARLEN]	The list of devices.

5.2.18 ABS_CALLBACK

```
void ABS_CALLBACK(
   IN const ABS_OPERATION *pOperation
   IN ABS_DWORD dwMsgID
   IN void *pMsgData
)
```

Description	A type of the callback function that an application can supply to the BSAPI to enable itself to display GUI state information to user. The callback is passed into the BSAPI function of interactive operations through ABS_OPERATION structure. Interactive operations call the callback repeatedly while the operation is in process. Thus the application can react accordingly to the process stage of the operation and update user interface. Note that exact way when the callback is called can be further determined by member Flags of ABS_OPERATION. Most applications will probably implement a callback in a way that it will create a dialog when first message of a biometric operation appears and then update a text and/or image in the dialog, according to the messages received. For this reason, BSAPI.DLL delays sending of some messages if there would be danger that the letter would replace the former too quickly so that end user would have no chance to catch the message. For example when user swipes incorrectly, and bad quality of the swipe or of resulted image is detected, the callback is called with appropriate feedback message. Then there is some delay before the callback is called again, with a prompt for new swipe so user has time to see the bad quality feddback. However this default behavior might not be desired in some other scenarios. For example if the callback implementation writes every message to a new line, so user can review complete history of the messages, or when the callback implementation provides no feedback to the user. In these cases the delays between some subsequent messages only protract time of the bioemtric operation. To disable all those delays, set the flag ABS_OPERATION_FLAG_LL_CALLBACK in ABS_OPERATION::Flags. The second supported flag of ABS_OPERATION related to ABS_CALLBACK is ABS_OPERATION_FLAG_USE_IDLE. When set, BSAPI.DLL guarantees the callback is called quite often (about 100 milliseconds). If there is nothing it would report, it uses message ABS_MSG_IDLE. The only purpose of this is to allow to cancel the operation from the callback in a reasonable way (see ABSCancelOperation for more info). However this comes at some cost: it eats more CPU cycles, so when this is not needed (e.g. you know that you don't cancel the operation or when you can cancel it from other thread), you should avoid use of this flag. When the flag ABS_OPERATION_FLAG_USE_IDLE is not set, the message ABS_MSG_IDLE is never used and it there might be quite a long time between two subsequent calls of the callback, e.g. when the biometric operation has been started but user does not touch the sensor for long time.	
Parameters		
pOperation		Pointer to ABS_OPERATION structure used when calling the interactive operation.

	The caller of the interactive operation can use member Context of the structure to pass data into the callback.
dwMsgID	ID of message. See description of ABS_MSG_xxxx constants.
pMsgData	Pointer to data with additional information related with the message. Its meaning is message dependent. Refer to documentation of specific ABS_MSG_xxxx constants.

6 Specific Constants

6.1 Flags for ABSInitializeEx (ABS_INIT_FLAG_xxxx)

The following flags can be used in function ABSInitializeEx.

ABS_INIT_FLAG_NT_SERVICE	0x1
Initializes the library in a mode compatible with Windows NT service. This mode is supported only on MS Windows. You should use this flag only when your application is running as NT service. Note that this flag cannot be used together with ABS_INIT_FLAG_FORCE_REMOTE_SENSOR. When flag ABS_INIT_FLAG_NT_SERVICE is used, only local devices can be opened, regardless whether ABS_INIT_FLAG_FORCE_LOCAL_SENSOR flag is or is not used.	
ABS_INIT_FLAG_FORCE_LOCAL_SENSOR	0x2
Forces BSAPI to ignore remote sessions and always open sensors locally. This flag cannot be used together with ABS_INIT_FLAG_FORCE_REMOTE_SENSOR. This mode is supported only on MS Windows.	
ABS_INIT_FLAG_FORCE_REMOTE_SENSOR	0x4
Forces BSAPI to open sensors in a remote session. I.e. if the session is not remote (via Terminal Services or Citrix), no device can be opened. This mode cannot be used in a service, i.e. it cannot be used together with ABS_INIT_FLAG_NT_SERVICE flag. Also it cannot be used toggether with ABS_INIT_FLAG_FORCE_LOCAL_SENSOR. This mode is supported only on MS Windows.	

6.2 Flags for ABS_OPERATION (ABS_OPERATION_FLAG_xxxx)

The following flags can be used in structure ABS_OPERATION.

ABS_OPERATION_FLAG_LL_CALLBACK	0x1
Enables low level callback mode. See documentation of ABS_CALLBACK for more information how the low level callback mode differs from the default high level mode.	
ABS_OPERATION_FLAG_USE_IDLE	0x2
Enables sending of messages ABS_MSG_IDLE to operation callback. By default idle messages are not sent. If hey are enabled, they are called in short intervals so you can call AB-SCancelOperation effectively from the operation callback. You should not allow sending the idle messages unless you really need them.	

6.3 Flags for Biometric and Image Grabbing Functions (ABS_FLAG_xxxx)

The following flags can be used for biometric functions.

Note that not all functions accept all flags. See documentation of respective function for more information.

ABS_FLAG_NOTIFICATION	0x1
Enables notification mode of the biometric operation. In the notification mode the biometric function does not provide any feedback to user until the user swipes. This is useful for applications running on background, when it's not desired to disturb user until he swipes. Whether the GUI dialog should be visible or not is controlled by messages ABS_MSG_DLG_SHOW and ABS_MSG_DLG_HIDE. Only functions ABSCapture and ABSVerify support this flag.	
ABS_FLAG_AUTOREPEAT	0x2
Enables auto-repeat mode of the biometric operation. If used, the verification is automatically restarted when the user's swipe does not match any template in a provided template set. Only function ABSVerify supports this flag. See documentation of this function for more details.	
ABS_FLAG_STRICT_PROFILE	0x4
Requires strict interpretation of raw grab profile. When set and any of the requested profile data cannot be respected because the FM does not support it, the raw grab operation fails and ANS_STATUS_NOT_SUPPORTED is returned. When not set the profile is followed only to degree supported by the device. I.e. those not supported are silently ignored, and the operation continues. Please note that profile key ABS_PKEY_IMAGE_FORMAT is always interpreted in the strict way. Only function ABSRawGrab supports this flag.	
ABS_FLAG_HIGH_RESOLUTION	0x8
Requires to use high sample image resolution. Some devices do not use high resolution by default because it has some huge impact on performance, and amount of data sent from the device. Using ABSGrab with this flag is device independent alternative to ABSRawGrab with exact device format specified, but that requires the caller to choose the right format depending device type.	

6.4 Template Purpose Constants (ABS_PURPOSE_xxxx)

Possible values used where purpose of fingerprint template (BIR)

Biometric functions which take purpose as one of their parameter can use this information to optimize operation processing. For example enrollment usually requires a higher template quality, so built-in biometric tests for template quality are stricter when ABS_PURPOSE_ENROLL is specified.

Please notice that the defined constants correspond to constants defined in BioAPI (BioAPI_PURPOSE_xxxx). However also note that BSAPI supports uses only subset of the purposes supported defined in BioAPI.

ABS_PURPOSE_UNDEFINED The purpose is not specified. The biometric operation is not optimized for any particular BIR purpose.	0
ABS_PURPOSE_VERIFY BIR is intended to be used for verification.	1
ABS_PURPOSE_ENROLL BIR is intended to be used for enrollment.	3

6.5 Key Constants for ABS_PROFILE_DATA (ABS_PKEY_xxxx)

Constants ABS_PKEY_xxxx are possible values for member Key of

ABS_PKEY_WAIT_FOR_ACCEPTABLE	1
Enables or disables waiting for acceptable scan quality. When set to non-zero, the waiting is enabled. When set to zero, it is disabled. Note that the setting has slightly different effect on devices depending whether hey have area or strip sensor. If the waiting is enabled then devices equiped with area sensor scan repeatedly until something resembling real finger is scanned (or until the operation is stopped due timeout or canceled or other error occures.) If disabled the area sensor just scans once regardless if any finger is present or not. By default, the waiting for acceptable is enabled for area sensor devices. If the waiting is enabled then devices with strip sensors can ask for multiple swipes, until the scanned image meets all required quality checks (as set with ABS_PKEY_SCAN_QUALITY_QUERY and ABS_PKEY_IMAGE_QUALITY_QUERY). If disabled, the raw grab operation requires only one finger swipe. By default, the wait for acceptable is disabled for the strip sensor devices. Note that when quality checks are disabled (i.e. both ABS_PKEY_SCAN_QUALITY_QUERY and ABS_PKEY_IMAGE_QUALITY_QUERY are set to non-zero), this flag has no effect for strip sensors because any swipe is then considered as acceptable.	
ABS_PKEY_SCAN_QUALITY_QUERY	2
Sets scan quality check mode. When set to non-zero (default), scanning quality problems are ignored during the swipe and thus they are not sent to callback. You can still retrieve some information about scan quality in form of ABS_SWIPE_INFO structure. When set to zero, scanning quality problems are sent to the callback in form of ABS_MSG_QUALITY_xxxx messages.	
ABS_PKEY_IMAGE_QUALITY_QUERY	3
Sets image quality check mode. When set to non-zero (default), image quality problems are ignored during the swipe and thus they are not sent to callback. You can still retrieve some information about scan quality in form of ABS_SWIPE_INFO structure. When set to zero, image quality problems are sent to the callback in form of ABS_MSG_QUALITY_xxxx messages.	
ABS_PKEY_ALLOW_HW_SLEEP	4
Enables HW sleep mode. When set to non-zero (default), HW sleep of the device is enabled during the raw grab operation. When set to zero, the sleep is disabled.	

ABS_PKEY_IMAGE_FORMAT	5
Specifies desired image format. Value can be any ABS_PVAL_IFMT_xxxx constant. Note that various devices support different image formats. Using unsupported image format causes ABSRawGrab to fail with ABS_STAUS_NOT_SUPPORTED, regardless whether strict mode is used or not.	
ABS_PKEY_REC_TERMINATION_POLICY	6
Specifies image reconstruction termination policy. It can be any ABS_PVAL_RTP_xxxx constant. It determines when the FM stops to scan the finger. Default value depends on the FM model used: • TFM 2.0: ABS_PVAL_RTP_CORE • ESS 2.1: ABS_PVAL_RTP_CORE • ESS 2.2: ABS_PVAL_RTP_CORE_PLUS • SONLY: ABS_PVAL_RTP_CORE_PLUS • TCD 50: ABS_PVAL_RTP_FINGERTIP	
ABS_PKEY_REC_RETUNING	7
Enables automatic sensor retuning. When set to non-zero (default), an automatic sensor calibration tuning is enabled while waiting for finger in order to always get the best image. When set to zero, the calibration tuning is disabled.	
ABS_PKEY_REC_DIGITAL_GAIN	8
This value is used for digital image enhancement. The value determines a factor of digital image enhancement. It is strongly recommended not to change this parameter. Supported only by TFM 2.0 and ESS 2.1.	
ABS_PKEY_REC_FLAG_DGAIN	9
This value is used for digital image enhancement. When set to non-zero (default), digital gain enhancement is enabled; when zero, it is disabled. Supported only by TCD 50	
ABS_PKEY_REC_FLAG_SRA_DOWN	10
Enables top-down striation removal algorithm.	

When set to non-zero (default), the algorithm is enabled. When zero, it is disabled.	
ABS_PKEY_REC_FLAG_SRA_UP Enables bottom-up striation removal algorithm. When set to non-zero (default), the algorithm is enabled. When zero, it is disabled.	11
ABS_PKEY_REC_FLAG_SKEW Enables skew compensation algorithm. When set to non-zero, the algorithm is enabled. When zero, it is disabled. Supported only by TCD 50.	12
ABS_PKEY_REC_FLAG_GRADIENT Enables gradient compensation algorithm. When set to non-zero, the algorithm is enabled. When zero, it is disabled. Supported only by TCD 50.	13
ABS_PKEY_REC_SWIPE_DIRECTION Specifies swipe direction mode. Can be set to any ABS_PVAL_SWIPEDIR_xxxx constant. The default value is ABS_PVAL_SWIPEDIR_STANDARD. If you set it to any non-default value you should set ABS_PKEY_SCAN_QUALITY_QUERY to zero as well. Not supported by TFM 2.0 and ESS 2.1.	14
ABS_PKEY_REC_NOISE_ROBUSTNESS Specifies noise robustness mode. Can be set to any ABS_PVAL_NOIR_xxxx constant. Default is ABS_PVAL_NOIR_DISABLED for SONLY and ABS_PVAL_NOIR_ON_DETECTION for TCD 50. Supported only by SONLY and TCD 50.	15
ABS_PKEY_REC_NOISE_ROBUSTNESS_TRIGGER Specifies noise robustness trigger. It determines how many consecutive bad swipes triggers noise robustness. Zero means no triggering by bad swipes. Default value is 3. Supported by TCD 50 only.	16
ABS_PKEY_REC_SWIPE_TIMEOUT Timeout for swipe termination in milliseconds. If this timeout expires, image reconstruction is terminated. Still the image reconstructed so far is passed to next processing which decides about its quality.	17

Default timeout is 6000 ms. Not supported by TFM 2.0 and ESS 2.1.	
ABS_PKEY_REC_NO_MOVEMENT_TIMEOUT	18
No movement timeout. If no movement is detected for that period (in milliseconds), the swipe is terminated regardless on the finger presence. This feature is disabled if set to zero. Default is 500 ms. Not supported by TFM 2.0 and ESS 2.1.	
ABS_PKEY_REC_NO_MOVEMENT_RESET_TIMEOUT	19
No movement reset timeout. If no movement is detected for that period (in milliseconds) and image is very short, the reconstruction is not restarted any more. This feature is disabled if set to zero. Default is 2000 ms. Not supported by TFM 2.0 and ESS 2.1.	
ABS_PKEY_SENSOR_SECURITY_MODE	20
Sensor security mode. Some sensors support HW encryption of image data being scanned. This parameter specifies if this encryption is used. It can be set to any ABS_PVAL_SSM_xxxx constant. The default value depends on device used, and on setting of global parameter ABS_PARAM_SENSOR_SECURITY. Supported only by SONLY and TCD 50.	
ABS_PKEY_DETECT_LATENT	21
Anti-latent checking mode. This parameter sets whether anti-latent checking is performed during the grab operation. When set to zero, only fast anti-latent check is performed, when set to 1, full (and slower) anti-latent check is performed. Default value is device dependent. If the scanned image is evaluated as being a latent image of a finger scanned previously, then the callback function gets message ABS_MSG_QUALITY_TOO_LIGHT. This settings has an effect only for area sensors. Strip sensors ignore it.	
ABS_PKEY_READER_SECURITY_MODE	22

Reader security mode. Some devices support encryption of data afrer reading data from sensor and reading from non-volatile device memory, so only strongly encrypted data are then sent between the fingerprint device and computer. It can be set to any ABS_PVAL_RSM_xxxx constant. The default value depends on device used, and on setting of global parameter ABS_PARAM_SENSOR_SECURITY. Supported only by SONLY and TCD 50.

6.6 ABS_PKEY_IMAGE_FORMAT Values (ABS_PVAL_IFMT_xxxx)

Possible image formats.

Please note that the desired image format is always evaluated in a strict mode of raw grab profile.

Note that the symbolic constant names contain some marginal parameters of the desired format: horizontal and vertical resolution in dots per inch (DPI), bits per pixel (they determine color count of the resulted sample image).

Remember that the structure ABS_IMAGE always uses one byte per pixel, regardless of the desired image format. (The image uses more compact representation during communication between the FM device and computer.)

Some image format contain word BINARIZED in the symbolic constant name. In this case the image is binarized yet on the device (with the exception of SONLY), so that the communication is faster.

Note that support for the particular image format depends on exact firmware version and whether the device was calibrated for the image format. Especially low power modes may require calibration. Therefore the information in the table below what FM models support which format is only for basic orientation.

ABS_PVAL_IFMT_381_381_8 Finger is grabbed with 3:4 subsampling (every 4 pixels scaled down to 3 pixels), 381 x 381 DPI, 8 bits/pixel. Supported by TFM, ESS, SONLY, TCD50v1, TCD50v2, TCD50v3.	2
ABS_PVAL_IFMT_254_254_8 Finger is grabbed with 1:2 subsampling (every second pixel), 254 x 254 DPI, 8 bits/pixel. Supported by TFM, ESS.	3
ABS_PVAL_IFMT_381_381_8_BINARIZED Finger is grabbed 3:4 subsampling (every 4 pixels scaled down to 3 pixels), 381 x 381 DPI, 8 bits/pixel and binarized to 1 bit/pixel. Supported by TFM, ESS, SONLY, TCD50v1, TCD50v2, TCD50v3.	4
ABS_PVAL_IFMT_508_254_8 Grab the whole finger with 1:2 subsampling in Y axis (508 x 254 DPI), 8 bits per pixel. Supported by ESS, TCD50v1 (some firmware variants only).	5
ABS_PVAL_IFMT_508_508_4 Grab the whole finger in full resolution (508 DPI, 8 bits), and cut off to 4 bits per pixel. Supported by ESS, TCD50v1.	6
ABS_PVAL_IFMT_381_381_4 Grab the whole finger with 3:4 subsampling (381 x 381 DPI, 8 bits), and cut off to 4 bits per pixel. Supported by ESS, TCD50v1, TCD50v2.	7
ABS_PVAL_IFMT_508_254_4 Grab the whole finger with 1:2 subsampling in Y axis (508 x 254 DPI, 8 bits), and cut off to 4 bits per pixel. Supported by ESS, TCD50v1 (some firmware variants only).	8

ABS_PVAL_IFMT_254_254_4 Grab the whole finger with 1:2 subsampling (254 x 254 DPI, 8 bits), and cut off to 4 bits per pixel. Supported by ESS.	9
ABS_PVAL_IFMT_508_508_8_WIDTH208 Grabs a centered windows of size 208 x 288 pixels, in full resolution of 508 x 508 DPI and 8 bits/pixel. Supported by TFM, ESS 2.2.	10
ABS_PVAL_IFMT_508_508_8_COMPRESS1 Grab the whole finger in full resolution (508 DPI), 8 bits per pixel. As ESS currently does not have enough RAM to hold the image of the whole fingerprint, the image is on-the-fly compressed using a lossy algorithm and transferred to the host in this format. The BSAPI.DLL library takes care of uncompressing the image. It is strongly recommended to use ABS_PVAL_IFMT_508_508_8_COMPRESS2 format in ESS2.2 instead of COMPRESS1. Supported by ESS.	11
ABS_PVAL_IFMT_508_508_4_SCAN4 Grab the whole finger in full resolution (508 DPI) in 4-bit scanning mode. This mode has lower power consumption but also lower image quality. Supported by ESS 2.1 older then rev.K.	12
ABS_PVAL_IFMT_381_381_8_FAST Grab the whole finger with 3:4 subsampling (381 x 381 DPI), 8 bits/pixel. This mode internally uses the 508 x 254 scanning, which supports faster finger movements at the cost of lower image quality. Supported by ESS, TCD50v1 (some firmware variants only).	13
ABS_PVAL_IFMT_508_254_4_SCAN4 Grab the whole finger with 1:2 subsampling in Y axis (508 x 254 DPI) in 4-bit scanning mode. This mode has lower power consumption but also lower image quality. Supported by ESS 2.1 older then rev.K.	14
ABS_PVAL_IFMT_254_254_4_SCAN4 Grab the whole finger with 1:2 subsampling (254 x 254 DPI) in 4-bit scanning mode. This mode has lower power consumption but also lower image quality. Supported by ESS 2.1 older then rev.K.	15
ABS_PVAL_IFMT_381_381_4_FAST Grab the whole finger with 3:4 subsampling (381 x 381 DPI, 8 bits), and cut off to 4 bits/pixel. This mode internally uses the 508 x 254 scanning, which supports faster finger movements at the cost of lower image quality. Supported by ESS, TCD50v1 (some firmware variants only).	16
ABS_PVAL_IFMT_381_381_8_BINARIZED_FAST	17

Grab the whole finger with 3:4 subsampling (381 x 381 DPI, 8 bits), and binarize to 1 bit/pixel. This mode internally uses the 508 x 254 scanning, which supports faster finger movements at the cost of lower image quality. Supported by ESS, TCD50v1 (some firmware variants only).	
ABS_PVAL_IFMT_508_508_8_COMPRESS2 Grab the whole finger in full resolution (508 DPI), 8 bits per pixel. As ESS currently does not have enough RAM to hold the image of the whole fingerprint, the image is on-the-fly compressed using a lossy algorithm and transferred to the host in this format. The BSAPI.DLL library takes care of uncompressing the image. This format better manages image background color and digital gain information than ABS_PVAL_IFMT_508_508_8_COMPRESS1 format. Supported by ESS 2.2.	18
ABS_PVAL_IFMT_381_381_8_SCAN381 Grab the whole finger with 3:4 subsampling (every 4 pixels scaled down to 3 pixels, 381 x 381 DPI), 8 bits/pixel. Subsampling is done directly by the sensor using its native 381 scanning format. Supported by ESS 2.2 with TCS3C, SONLY, TCD50v1, TCD50v2, TCD50v3.	19
ABS_PVAL_IFMT_381_381_4_SCAN381 Grab the whole finger with 3:4 subsampling (381 x 381 DPI, 8 bits), and cut off to 4 bits per pixel. Subsampling is done directly by the sensor using its native 381 scanning format. Supported by ESS 2.2 with TCS3C, TCD50v1, TCD50v2.	20
ABS_PVAL_IFMT_381_381_8_BINARIZED_SCAN381 Grab the whole finger with 3:4 subsampling (every 4 pixels scaled down to 3 pixels, 381 x 381 DPI), 8 bits/pixel, and binarize to 1 bit/pixel. Subsampling is done directly by the sensor using its native 381 scanning format. Supported by ESS 2.2 with TCS3C, SONLY, TCD50v1, TCD50v2.	21
ABS_PVAL_IFMT_381_381_8_LP Grab the whole finger with 3:4 subsampling (every 4 pixels scaled down to 3 pixels, 381 x 381 DPI), 8 bits/pixel, uses low power consumption mode. Supported by ESS 2.2, TCD50v1 (only some special non-standard firmware variants).	22
ABS_PVAL_IFMT_381_381_4_LP Grab the whole finger with 3:4 subsampling (381 x 381 DPI, 8 bits), and cut off to 4 bits per pixel, uses low power consumption mode. Supported by ESS 2.2, TCD50v1 (only some special non-standard firmware variants).	23
ABS_PVAL_IFMT_381_381_8_BINARIZED_LP Grab the whole finger with 3:4 subsampling (every 4 pixels scaled down to 3 pixels, 381 x 381 DPI), 8 bits/pixel, and binarize to 1 bit/pixel, uses low power consumption mode. Supported by ESS 2.2, TCD50v1 (only some special non-standard firmware variants).	24
ABS_PVAL_IFMT_381_381_8_VLP	25

Grab the whole finger with 3:4 subsampling (every 4 pixels scaled down to 3 pixels, 381 x 381 DPI), 8 bits/pixel, uses very low power consumption mode. Supported by ESS 2.2 with serial communication not faster then 57600 kbps.	
ABS_PVAL_IFMT_381_381_4_VLP Grab the whole finger with 3:4 subsampling (381 x 381 DPI, 8 bits), and cut off to 4 bits per pixel, uses very low power consumption mode. Supported by ESS 2.2 with serial communication not faster then 57600 kbps.	26
ABS_PVAL_IFMT_381_381_8_BINARIZED_VLP Grab the whole finger with 3:4 subsampling (every 4 pixels scaled down to 3 pixels, 381 x 381 DPI), 8 bits/pixel, and binarize to 1 bit/pixel, uses very low power consumption mode. Supported by ESS 2.2 with serial connection not faster then 57600 kbps.	27
ABS_PVAL_IFMT_381_381_8_SCAN381_381_4 Grab the whole finger with 3:4 subsampling (every 4 pixels scaled down to 3 pixels, 381 x 381 DPI), 8 bits/pixel. Image is internally scanned using format 381/381/4. Supported only by some variants of SONLY.	28
ABS_PVAL_IFMT_381_381_8_BINARIZED_SCAN381_381_4 Grab the whole finger with 3:4 subsampling (every 4 pixels scaled down to 3 pixels, 381 x 381 DPI), 8 bits/pixel. Image is internally scanned using format 381/381/4. Supported only by some variants of SONLY.	30
ABS_PVAL_IFMT_381_381_8_SCAN381_254_4 Grab the whole finger with 3:4 subsampling (every 4 pixels scaled down to 3 pixels, 381 x 381 DPI), 8 bits/pixel. Image is internally scanned using format 381/254/4. Supported only by some variants of SONLY.	31
ABS_PVAL_IFMT_381_381_8_BINARIZED_SCAN381_254_4 Grab the whole finger with 3:4 subsampling (every 4 pixels scaled down to 3 pixels, 381 x 381 DPI), 8 bits/pixel, and binarize to 1 bit/pixel. Image is internally scanned using format 381/254/4. Supported only by some variants of SONLY.	33
ABS_PVAL_IFMT_508_508_8_SCAN508_508_8 Grab the whole finger in full resolution (508 x 508 DPI), 8 bits/pixel. Image is internally scanned using format 508/508/8. Supported only by some variants of SONLY, TCD50v3.	34
ABS_PVAL_IFMT_508_508_4_SCAN508_508_8 Grab the whole finger in full resolution (508 x 508 DPI), 8 bits/pixel, and cut off to 4 bits per pixel. Image is internally scanned using format 508/508/8. Supported only by some variants of SONLY, TCD50v1.	35

ABS_PVAL_IFMT_508_508_8_BINARIZED_SCAN508_508_8	36	Grab the whole finger in full resolution (508 x 508 DPI), 8 bits/pixel, and binarize to 1 bit/pixel. Image is internally scanned using format 508/508/8. Supported by some variants of SONLY.
ABS_PVAL_IFMT_508_508_8_COMPRESS3	39	Grab the whole finger in full resolution (508 x 508 DPI), 8 bits/pixel. As TCD50 currently does not have enough RAM to hold the image of the whole fingerprint, the image is on-the-fly compressed using a lossy algorithm and transferred to the host in this format. The BSAPI.DLL library takes care of uncompressing the image. Supported by TCD50v1, TCD50v2.
ABS_PVAL_IFMT_508_254_8_LP	40	Grab the whole finger with 1:2 subsampling in Y axis (508 x 254 DPI), 8 bits per pixel, uses low power consumption mode. Supported by TCD50v1 (some firmware variants only).
ABS_PVAL_IFMT_508_254_4_LP	41	Grab the whole finger with 1:2 subsampling in Y axis (508 x 254 DPI), 4 bits per pixel, uses low power consumption mode. Supported by TCD50v1 (some firmware variants only).
ABS_PVAL_IFMT_381_381_8_FAST_LP	42	Grab the whole finger with 3:4 subsampling (381 x 381 DPI), 8 bits/pixel, uses low power consumption mode with support for scanning of faster finger movements. Supported by TCD50v1 (some firmware variants only).
ABS_PVAL_IFMT_381_381_4_FAST_LP	43	Grab the whole finger with 3:4 subsampling (381 x 381 DPI), 4 bits/pixel, uses low power consumption mode with support for scanning of faster finger movements. Supported by TCD50v1 (some firmware variants only).
ABS_PVAL_IFMT_381_381_8_BINARIZED_FAST_LP	44	Grab the whole finger with 3:4 subsampling (381 x 381 DPI), 8 bits/pixel, and binarize to 1 bit/pixel, uses low power consumption mode with support for scanning of faster finger movements. Supported by TCD50v1 (some firmware variants only).

6.7 ABS_PKEY_REC_TERMINATION_POLICY Values (ABS_PVAL_RTP_xxxx)

See description of ABS_PKEY_REC_TERMINATION_POLICY for more information.

ABS_PVAL_RTP_BASIC Basic image reconstruction termination policy. If the scanned image would be longer then maximal allowed length, only beginning of the image from the start on is returned.	0
ABS_PVAL_RTP_FINGERTIP Fingertip image reconstruction termination policy. If the scanned image would be longer then maximal allowed length, the end of the image up to the fingertip is returned.	1
ABS_PVAL_RTP_CORE Core image reconstruction termination policy. If the scanned image would be longer then maximal allowed length, the most valuable part of the image from biometrical viewpoint (typically the fingerprint's core) is returned.	2
ABS_PVAL_RTP_CORE_PLUS Enhanced core image reconstruction termination policy. If the scanned image would be longer then maximal allowed length, the most valuable part of the image from biometrical viewpoint (typically the fingerprint's core), with finger joint skipped, is returned. Not supported by TFM 2.0 and ESS 2.1 .	3

6.8 ABS_PKEY_REC_SWIPE_DIRECTION Values (ABS_PVAL_SWIPEDIR_xxxx)

See description of ABS_PKEY_REC_SWIPE_DIRECTION for more information.

ABS_PVAL_SWIPEDIR_STANDARD Standard swipe direction.	0
ABS_PVAL_SWIPEDIR_INVERTED Inverted swipe direction.	1
ABS_PVAL_SWIPEDIR_AUTODETECT Autodetection at the beginning of the swipe.	2
ABS_PVAL_SWIPEDIR_STANDARD_WARN Standard swipe direction with warning. If backward swipe is detected, message ABS_MSG_QUALITY_BACKWARD is sent to callback. Note that ABS_PKEY_SCAN_QUALITY_QUERY must be set to 0 if this should work reliably.	3
ABS_PVAL_SWIPEDIR_INVERTED_WARN Inverted swipe direction with warning. If backward swipe is detected, message ABS_MSG_QUALITY_BACKWARD is sent to callback. Note that ABS_PKEY_SCAN_QUALITY_QUERY must be set to 0 if this should work reliably.	4

6.9 ABS_PKEY_REC_NOISE_ROBUSTNESS Values (ABS_PVAL_NOIR_xxxx)

See description of ABS_PKEY_REC_NOISE_ROBUSTNESS for more information.

ABS_PKEY_NOIR_DISABLED Noise robustness is switched off.	0
ABS_PKEY_NOIR_FORCED Noise robustness is switched on.	1
ABS_PKEY_NOIR_ON_DETECTION Noise robustness is in auto detection mode.	2

6.10 ABS_PKEY_SENSOR_SECURITY_MODE values (ABS_PVAL_SSM_xxxx)

See description of ABS_PKEY_SENSOR_SECURITY_MODE for more information.

ABS_PVAL_SSM_DISABLED Sensor security mode is disabled.	0
ABS_PVAL_SSM_ENCRYPT Sensor security is set to 'encryption' mode.	1
ABS_PVAL_SSM_SIGN_ALL Sensor security is set to 'sign all' mode.	2
ABS_PVAL_SSM_SIGN_PARTIAL_V1 Sensor security is set to 'sign partial ver. 1'. It is faster then version 2, but less secure.	3
ABS_PVAL_SSM_SIGN_PARTIAL_V2 Sensor security is set to 'sign partial ver. 2'. It is slower then version 1, but more secure.	4

6.11 ABS_PKEY_READER_SECURITY_MODE values (ABS_PVAL_RSM_xxxx)

See description of ABS_PKEY_SENSOR_SECURITY_MODE for more information.

ABS_PVAL_RSM_DISABLED Reader security mode is disabled.	0
ABS_PVAL_RSM_ENCRYPT Reader security is set to 'encryption' mode.	1

6.12 Swipe Info Flags (ABS_SWIPE_FLAG_xxxx)

Member Flags of structure ABS_SWIPE_INFO is a bitmask describing

ABS_SWIPE_FLAG_TOO_FAST The swipe was too fast. If user swipes too fast, the device is not able to process all the data.	0x01
ABS_SWIPE_FLAG_TOO_SKEWED The swipe was too skewed.	0x02
ABS_SWIPE_FLAG_BACKWARDS_MOVEMENT Swipe was in wrong direction. Note that this flag is set only in autodetection mode of the swipe direction (i.e. when profile value ABS_PKEY_REC_SWIPE_DIRECTION is set to ABS_PVAL_SWIPEDIR_AUTODETECT).	0x04
ABS_SWIPE_FLAG_JOINT_DETECTED Finger joint was detected in the swipe.	0x08
ABS_SWIPE_FLAG_TOO_SHORT The swipe was too short. It may cause low level quality of resulting biometric templates, because there only few biometric data in the swiped region of user's finger.	0x10
ABS_SWIPE_FLAG_TOO_LIGHT The swipe was too light. In case of area sensor this can also happen in a case of grabbed latentt grab.	0x20

6.13 Anti-latent Checking Flags (ABS_LATENT_xxxx)

Flags usable with functions ABSCheckLatent.

See documentation of function ABSCheckLatent and also chapter Anti-latent Checking for more information.

ABS_LATENT_OP_CHECK	0x01
Asks function ABSCheckLatent to perform anti-latent check.	
ABS_LATENT_OP_STORE	0x02
Asks function ABSCheckLatent to store the last scan for any subsequent checks.	

6.14 Process Constants (ABS_PROCESS_xxxx)

These constants identify interactive operation processes.

Structures ABS_PROCESS_DATA, ABS_PROCESS_BEGIN_DATA and ABS_PROCESS_SUCCESS_DATA have member called Process which identifies the stage the operation enters.

Some high level biometric operations consist of multiple processes. Generally the interactive operation is composed from tree of processes. Entering and leaving in node in the tree is marked with calling the callback with messages ABS_MSG_PROCESS_BEGIN and ABS_MSG_PROCESS_END.

Other ABS_MSG_PROCESS_xxxx may or may not be sent, depending on the respective process and its progress.

For example a typical consolidated enrollment operation can be composed from the sequence of the following messages:

1. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_ENROLL)
2. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_CONSOLIDATED_CAPTURE)
3. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_CAPTURE)
4. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_GRAB)
5. ... messages leading the user to correctly swipe his finger
6. ABS_MSG_PROCESS_END (end of ABS_PROCESS_GRAB)
7. ABS_MSG_PROCESS_PROGRESS (Percentage=23%)
8. ABS_MSG_PROCESS_END (end of ABS_PROCESS_CAPTURE)
9. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_CAPTURE)
10. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_GRAB)
11. ... messages leading the user to correctly swipe his finger
12. ABS_MSG_PROCESS_END (end of ABS_PROCESS_GRAB)
13. ABS_MSG_PROCESS_PROGRESS (Percentage=32%)
14. ABS_MSG_PROCESS_END (end of ABS_PROCESS_CAPTURE)
15. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_CAPTURE)
16. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_GRAB)
17. ... messages leading the user to correctly swipe his finger
18. ABS_MSG_PROCESS_END (end of ABS_PROCESS_GRAB)
19. ABS_MSG_PROCESS_PROGRESS (Percentage=39%)
20. ABS_MSG_PROCESS_END (end of ABS_PROCESS_CAPTURE)
21. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_CAPTURE)

22. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_GRAB)
23. ... messages leading the user to correctly swipe his finger
24. ABS_MSG_PROCESS_END (end of ABS_PROCESS_GRAB)
25. ABS_MSG_PROCESS_PROGRESS (Percentage=64%)
26. ABS_MSG_PROCESS_END (end of ABS_PROCESS_CAPTURE)
27. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_CAPTURE)
28. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_GRAB)
29. ... messages leading the user to correctly swipe his finger
30. ABS_MSG_PROCESS_END (end of ABS_PROCESS_GRAB)
31. ABS_MSG_PROCESS_PROGRESS (Percentage=88%)
32. ABS_MSG_PROCESS_END (end of ABS_PROCESS_CAPTURE)
33. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_CAPTURE)
34. ABS_MSG_PROCESS_BEGIN (ProcessID = ABS_PROCESS_GRAB)
35. ... messages leading the user to correctly swipe his finger
36. ABS_MSG_PROCESS_END (end of ABS_PROCESS_GRAB)
37. ABS_MSG_PROCESS_PROGRESS (Percentage=100%)
38. ABS_MSG_PROCESS_END (end of ABS_PROCESS_CAPTURE)
39. ABS_MSG_PROCESS_END (end of ABS_PROCESS_CONSOLIDATED_CAPTURE)
40. ABS_MSG_PROCESS_END (end of ABS_PROCESS_ENROLL)

ABS_PROCESS_NAVIGATE Root process of navigation (ABSNavigate). It typically consists of one subprocess of type ABS_PROCESS_CONSOLIDATED_CAPTURE.	1
ABS_PROCESS_ENROLL Root process of enrollment (ABSEnroll). It is typically composed of one subprocess ABS_PROCESS_CONSOLIDATED_CAPTURE.	2
ABS_PROCESS_VERIFY Root process of verification (ABSVerify) It has typically one subprocess of type ABS_PROCESS_CAPTURE.	3
ABS_PROCESS_IDENTIFY Root process of identification.	4
ABS_PROCESS_CONSOLIDATED_CAPTURE	5

Process of consolidated template from the scanner. It is a complex process, consisted of several subprocesses of ABS_PROCESS_CAPTURE and one final ABS_PROCESS_CONSOLIDATE.	
ABS_PROCESS_CONSOLIDATE Process of consolidation. It merges several templates of one finger into one high-quality enrollment template. Note that since BSAPI 3.5 this process is never used and the constant remains to be defined solely for backward compatibility, as source code of custom ABS_CALLBACK implementations could refer the constatnt.	6
ABS_PROCESS_CAPTURE Process of template capture from scanner. Typically it has one subprocess of type ABS_PROCESS_GRAB.	7
ABS_PROCESS_MATCH Process of matching template against set of templates.	8
ABS_PROCESS_GRAB Process of sample image grab from scanner. It's relatively low level process retrieving a fingerprint sample image from the sensor.	9
ABS_PROCESS_NOTIFY Process of notification. This process is used by functions which allow notification mode (see ABS_FLAGF_NOTIFICATION).	10

6.15 Device Property Constants (ABS_DEVPROP_xxxx)

The following constants are suitable as values for dwPropertyId parameter

ABS_DEVPROP_DEVICE_VERSION Identifies version of the FM device ROM. Output ABS_DATA contains 4 bytes, interpreted as ABS_DWORD. Highest byte specifies major version, second highest byte specifies minor version and low word specifies subversions/revisions. • 0x0200xxxx (2.0.x) - used by TFM 2.0 • 0x0401xxxx (4.1.x) - used by ESS 2.1 • 0x0402xxxx (4.2.x) - used by ESS 2.2 • 0x0500xxxx (5.0.x) - used by TCD 50 Note that SONLY (sensor only device without ROM) can return any number, denoting a version of the library built-in in BSAPI which emulates the device ROM. To distinguish chipset and SONLY devices, use bit 0x80000000 of ABS_DEVPROP_FUNCTIONALITY. See also parameter ABS_DEVPROP_SENSOR_TYPE.	0
ABS_DEVPROP_DEVICE_ID Unique identification of the device if the FM supports it. ABS_DATA can contain arbitrary sequence of bytes, composing the identification. If not supported by FM, no output ABS_DATA are allocated and NULL is passed out.	1
ABS_DEVPROP_FIRMWARE_VARIANT Identifies firmware variant. Output ABS_DATA contains 4 bytes, interpreted as ABS_DWORD.	2
ABS_DEVPROP_SENSOR_TYPE Identifies sensor type. Output ABS_DATA contains 4 bytes, interpreted as ABS_DWORD. The lowest byte specifies type of the sensor used in the fingerprint device. • 0x26 - area sensor TCS1C • 0x1B - area sensor TCS2C • 0x10 - swipe sensor TCS3B • 0x20 - swipe sensor TCS3C	4

• 0x32 - swipe sensor TCS4B • 0x33 - swipe sensor TCS4C • 0x36 - swipe sensor TCS4E • 0x43 - swipe sensor TCS5B • 0x45 - swipe sensor TCS4K Bit 0x80000000 specifies whether the sensor is swipe (also known as strip) sensor. When set the sensor is strip sensor, when not set the sensor is area sensor. All other bits which are not documented here are reserved, and their value is not defined.	
ABS_DEVPROP_SYSTEM_ID	6
System identification bits. The value is set in firmware of the device and can be used to identify some devices with some specific purpose. Most devices have this value set to zero. If supported (ESS 2.2 and newer), output ABS_DATA contains 4 bytes, interpreted as ABS_DWORD. If not supported no output ABS_DATA are allocated and NULL is passed out.	
ABS_DEVPROP_SYSTEM_NAME	7
System identification name. The value is set in firmware of the device and can be used to identify some devices with some specific purpose. If supported (ESS 2.2 and newer) and set, output ABS_DATA contains arbitrary count of bytes, interpreted as a null-terminated string. If not supported no output ABS_DATA are allocated and NULL is passed out.	
ABS_DEVPROP_FUNCTIONALITY	8
Provides information about FM capabilities. Output ABS_DATA contains 4 bytes, interpreted as bitmask. Bit 0x80000000 specifies whether the FM is SONLY (bit is set) or chipset-based FM (bit is unset).	
ABS_DEVPROP_DSN_STRING	11
Provides DSN of the device. Output ABS_DATA contains any number of bytes, last of them is always zero. Interpret them as zero-terminated C string. It can be used as DSN string for functions ABSOpen and ABSEnumerateDevices.	
ABS_DEVPROP_GUID	12
Gets GUID of the device. Output ABS_DATA contains a binary GUID stored on the device. The GUID is generated on first device boot or during loading firmware into NVM (depending on device type).	

Note that only ESS 2.2 and newer devices support this feature.	
ABS_DEVPROP_USAGE	13
Type of the reader in which the FM is used. Output ABS_DATA contains 4 bytes, interpreted as bitmask. • 0x80000000: If this bit is set, the sensor is internal. • 0x40000000: If this bit is set, the sensor is external. • 0x000000ff: The lowest byte specifies a constant which determines how device LEDs blink when the device is in automatic blinking mode (i.e. how the device blinks when the current LED mode is ABS_LED_MODE_AUTO).	

6.16 LED Blinking Mode Constants (ABS_LED_MODE_xxxx)

For more information about each mode, se the chapter Blinking with LEDs

ABS_LED_MODE_MANUAL Manual control of the LEDs.	0
ABS_LED_MODE_AUTO Automatic control of the LEDs.	1
ABS_LED_MODE_MANUAL2 Advanced manual control of the LEDs (TCS5D only).	3
ABS_LED_MODE_OFF LEDs are turned off.	0xFFFFFFFF

6.17 LED Bits for Flags in ABS_LED_PARAMS_MANUAL2 Structure (ABS_LEDDRV_FLAG_xxxx)

BS_LEDDRV_FLAG_START_ON	0x00000001
If set, state machine will start from the beginning of "ON" period.	
ABS_LEDDRV_FLAG_STOP_AFTER_OFF	0x00000002
If set, state machine will stop after first "OFF" period finishes.	
ABS_LEDDRV_FLAG_BR_INVERT	0x00000004
If set, brightness output is inverted.	
ABS_LEDDRV_FLAG_TABLE_COSEXP	0x00000008
If set, brightness level is defined as CosExp function, otherwise linear function is used.	

6.18 Session and Global Parameter Constants (ABS_PARAM_xxxx)

These constants identify each parameter and can be used as value of parameter dwParamID of ABSSetSessionParameter and ABSGetSessionParameter (for session parameters) functions, or ABSSetGlobalParameter and ABSGetGlobalParameter (for global parameters).

BSSRV.DLL (functions ABSSrvSetGlobalParameter and ABSSrvGetGlobalParameter) supports only parameters ABS_PARAM_MATCH_LEVEL and ABS_PARAM_IFACE_VERSION. The latter can be only read with ABSSrvGetGlobalParameter.

ABS_PARAM_CONSOLIDATION_COUNT	1
Obsolete. Use parameters ABS_PARAM_CONSOLIDATION_COUNT_MIN and ABS_PARAM_CONSOLIDATION_COUNT_MAX instead. This parameter is only kept for backward compatibility and new code should not use it. Setting this parameter to 3 or 5 is equivalent to setting both ABS_PARAM_CONSOLIDATION_COUNT_MIN and ABS_PARAM_CONSOLIDATION_COUNT_MAX to the same value. Setting this parameter to zero is equivalent to setting ABS_PARAM_CONSOLIDATION_COUNT_MIN to 3 and ABS_PARAM_CONSOLIDATION_COUNT_MAX to 10. When this parameter is read, then the returned value depends on actual values of ABS_PARAM_CONSOLIDATION_COUNT_MIN and ABS_PARAM_CONSOLIDATION_COUNT_MAX. If those other parameters are equal then that value is returned, If those parameters differ, then zero is returned.	
ABS_PARAM_CONSOLIDATION_TYPE	2
Determines type of consolidation. I.e. how multiple templates are mixed together, to get one high-quality template. See description of parameter ABS_PARAM_CONSOLIDATION_COUNT as well. Value is represented as ABS_DATA with 1 byte of length. The value can be any constant ABS_CONSOLIDATION_xxxx. See description of those constants.	
ABS_PARAM_MATCH_LEVEL	3
Determines required level of security for comparing two templates with ABSVerifyMatch. Value is represented as ABS_DATA with 1 byte of length. For the list of supported values, see ABS_MATCH_xxxx.	
ABS_PARAM_DISABLE_SENSOR_SLEEP	4
Allows to disable sensor sleeping. Value is represented as ABS_DATA with 1 byte of length. When zero, BSAPI can switch the device to sleep mode, to save power. When non-zero, the sleep mode is disabled.	
ABS_PARAM_DISABLE_SELECTIVE_SUSPEND	5
Allows to disable the selective suspend.	

Value is represented as ABS_DATA with 1 byte of length. When zero, the selective suspend can be used. When non-zero, the selective suspend is disabled.	
ABS_PARAM_POWER_SAVE_MODE	6
Sets default power safe mode, if it is not disabled completely with ABS_PARAM_DISABLE_SENSOR_SLEEP. Value is represented as ABS_DATA with 1 byte of length. Possible values are: 0 – power save is always off (high power consumption); 1 – power save is always on (minimal power consumption, higher time latencies can occur); 2 – power save switching depending on the user activity. See also ABS_PARAM_POWER_SAVE_TIMEOUT.	
ABS_PARAM_POWER_SAVE_TIMEOUT	7
Together with other parameters, it determines how power save works. This parameter specifies timeout since last user activity to allow selective suspend for the fingerprint device. It has some effect to power management only if ABS_PARAM_POWER_SAVE_MODE is set to 2. Value is represented as ABS_DATA with 4 bytes of length. The value is interpreted as ABS_DWORD, determining number of seconds. Default is 3 minutes. See also ABS_PARAM_POWER_SAVE_MODE, ABS_PARAM_POWER_SAVE_CHECK_KEYBOARD and ABS_PARAM_POWER_SAVE_ACTIVE_TO_SLEEP_TIMEOUT.	
ABS_PARAM_ANTISPOOFING_POLICY	8
Value is represented as ABS_DATA with 1 byte of length. For the list of supported values, see ABS_ANTISPOOFING_xxxx.	
ABS_PARAM_ANTISPOOFING_LEVEL	9
If antispoofing algorithms are applied, this settings determines trade-off between security and user's convenience. Value is represented as ABS_DATA with 1 byte of length. Possible values are: 0 - convenience is preferred (default); 1 - security is preferred.	
ABS_PARAM_OPEN_TOTAL_TIMEOUT	10
Total open-session timeout. When some specific error occurs with device (e.g. communication error caused by ESD), BSAPI automatically attempts to restore communication session with the device. This parameter specifies maximal amount of time BSAPI attempts to reopen the session. Value is represented as ABS_DATA with 4 bytes of length. The value is interpreted as ABS_DWORD, determining number of milliseconds. Default value is 5000.	
ABS_PARAM_OPEN_RETRY_UI_NOTIFY_TIMEOUT	11
Timeout to user-interface notification about the reopen attempt.	

		When BSAPI is attempting to reopen session for time longer then this this parameter specifies, then callback of an interactive operation receives message ABS_MSG_PROCESS_PROGRESS, so the end-user is notified that the device is busy. Value is represented as ABS_DATA with 4 bytes of length. The value is interpreted as ABS_DWORD, determining number of milliseconds. Default value is 2000.
ABS_PARAM_OPEN_RETRY_DELAY	12	Delay between two subsequent session reopen attempts. BSAPI attempts to reopen the session repeatedly until it succeeds or until ABS_PARAM_OPEN_TOTAL_TIMEOUT expires. This parameter then specifies delay between two subsequent reopen attempts. The value is represented as ABS_DATA with 4 bytes of length. The value is interpreted as ABS_DWORD, determining number of milliseconds. Default value is 500.
ABS_PARAM_IFACE_VERSION	13	Read-only global parameter which determines version of BSAPI interface. Value is represented as ABS_DATA with 1 byte of length. This version of BSAPI uses version 2 of the interface. Note that corresponding versions of BSAPI.DLL and BSSRV.DLL always get the same value of this parameter.
ABS_PARAM_POWER_SAVE_CHECK_KEYBOARD	14	This global parameter determines if keyboard and mouse are treated as an user activity. Value is represented as ABS_DATA with 4 bytes of length. Zero means the keyboard and mouse actions are not treated as an user activity, so only touching the sensor has impact to power management. Non-zero means the keyboard and mouse are considered an user activity. On Windows, the default value is 1 unless BSAPI is running in NT service compatible mode i.e. unless it is initialized with function ABSInitializeEx with flag ABS_INIT_FLAG_NT_SERVICE set. If BSAPI is in NT service compatible mode, the default value is zero. Note that on Windows and when the parameter is set to 1, the user activity is detected only in a context of active user's session. if the process does not run in active user's session, user actions on keyboard and mouse are not detected. On other systems, default value is zero and setting the value is not supported. See also ABS_PARAM_POWER_SAVE_MODE and ABS_PARAM_POWER_SAVE_TIMEOUT.
ABS_PARAM_LATENT_CHECK	16	This global parameter determines whether antilatent checks are performed implicitely.

Value is represented as ABS_DATA with 1 byte of length. If set to 0, the antiltent checks are disabled, when 1 (default) the cheks are enabled, so any call to ABSEnroll or ABSVerify implicitely checks for latent fingerprints on area sensors. After calling another functions (e.g. ABSGrab or ABSCapture), it's up on application to do the check manually with function ABSCheckLatent if it desires to do so. See chapter Anti-latent Checking for more information.	
ABS_PARAM_SENSOR_SECURITY This global parameter determines security level of communication between fingerprint device and computer. Value is represented as ABS_DATA with 1 byte of length. The value can be any constant ABS_SENSOR_SECURITY_xxxx. See description of those constants.	17
ABS_PARAM_POWER_SAVE_ACTIVE_TO_SLEEP_TIMEOUT Determines timeout since last user activity to enable sensor sleeping. After this timeout elapses, the sensor in fingerprint device is not powered until user places his finger on it. Value is represented as ABS_DATA with 4 bytes of length, and the timeout is measured in seconds. Default is 30 seconds. It is recommended this timeout should be shorter then then the one specified with ABS_PARAM_POWER_SAVE_TIMEOUT. See also ABS_PARAM_POWER_SAVE_MODE and ABS_PARAM_POWER_SAVE_TIMEOUT.	18
ABS_PARAM_POWER_SAVE_RESET_ON_OPERATION_START This global parameter specifies whether power save timeouts are reset on operation start. When set to 1, operation start is also considered to be an user activity event and user activity timeouts (namely ABS_PARAM_POWER_SAVE_TIMEOUT and ABS_PARAM_POWER_SAVE_ACTIVE_TO_SLEEP_TIMEOUT) are measured since operation start or specified user activity, whatever happens later. In other words, when set to 1, the device is in full-power mode when starting any operation with the fingerprint device regardless of user's last activity on the fingerprint sensor. When set to 0 (default), operation start has no influence on timeouts specified above.	19
ABS_PARAM_IGNORE_BIOMETRIC_TIMEOUT This global parameter determines whether biometric timeout reported by device is ignore or not. When in sleep mode, the device can still detect presense of user's finger on the bezel framing the sensor. When the finger is detected, the device wakes up the sensor and it starts scanning of the finger. However waking the sensor up can take few tens milliseconds and by that time the finger might be gone. If the woken sensor cannot detect a finger on it for some short time after the finger has been detected on the bezel, a biometric timeout is reported by the device.	20

This situation can also happen accidentaly, especially with sensors integrated to a body of notebook (typically quite close to a keyboard). Value is represented as ABS_DATA with 1 byte of length. If set to zero, the biometric timeout situation is translated to ABS_MSG_QUALITY_TOO_FAST. If set to 1 (default), there is no such GUI feedback and the as BSAPI assumes the biometric timeout situation is accidental. In both cases, the biometric operation continues and expects user to swipe his finger again.	
ABS_PARAM_CONSOLIDATION_COUNT_MIN	21
Specifies minimal count of finger swipes for consolidation. Consolidation is a process used for merging information acquired from multiple fingerprint template to one high-quality template, used in enrollment operation. This parameter determines minimal count of how many templates are consolidated into one enrollment template. The value is represented as ABS_DATA with 1 byte of length. Only values in the range 3 - 10 are allowed. Default value is 3. Note that behavior of enrollment operation is undefined if ABS_PARAM_CONSOLIDATION_COUNT_MIN is greater then ABS_PARAM_CONSOLIDATION_COUNT_MAX.	
ABS_PARAM_CONSOLIDATION_COUNT_MAX	22
Specifies maximal count of finger swipes for consolidation. Consolidation is a process used for merging information acquired from multiple fingerprint template to one high-quality template, used in enrollment operation. This parameter determines maximal count of how many templates are consolidated into one enrollment template. The value is represented as ABS_DATA with 1 byte of length. Only values in the range 3 - 10 are allowed. Default value is 10. Note that behavior of enrollment operation is undefined if ABS_PARAM_CONSOLIDATION_COUNT_MAX is lower then ABS_PARAM_CONSOLIDATION_COUNT_MIN.	

6.19 Parameter ABS_PARAM_CONSOLIDATION_TYPE Values (ABS_CONSOLIDATION_xxxx)

These constants are intended as possible values of parameter

ABS_CONSOLIDATION_NORMAL Normal consolidation algorithm is used. Enrollment template is constructed either from a subset of collected templates, or it uses one (the best) of provided templates. A built-in heuristic makes the decision which one of the approaches is used.	0
ABS_CONSOLIDATION_CONVENIENT Convenient consolidation algorithm is used. Similar to ABS_CONSOLIDATION_NORMAL policy with relaxed criteria for image/template acceptance for entering the enrollment process.	1
ABS_CONSOLIDATION_STRICT Strict consolidation algorithm is used. Similar to ABS_CONSOLIDATION_NORMAL policy, except that all collected templates must match each other (i.e. the same finger has to be used for all acquisitions).	2

6.20 Parameter ABS_PARAM_MATCH_LEVEL Values (ABS_MATCH_xxxx)

These constants are intended as possible values of parameter

ABS_MATCH_MIN_SECURITY Minimal security setting.	1
ABS_MATCH_LOWER_SECURITY Lower security setting.	2
ABS_MATCH_MEDIUM_SECURITY Medium security setting.	3
ABS_MATCH_HIGHER_SECURITY Higher security setting.	4
ABS_MATCH_MAX_SECURITY Maximal security setting.	5

6.21 Parameter ABS_PARAM_ANTISPOOFING_POLICY Values (ABS_ANTISPOOFING_xxxx)

These constants are intended as possible values of parameter

ABS_ANTISPOOFING_DISABLED Antispoofing checks are explicitly turned off on the fingerprint sensor. This is default value.	0
ABS_ANTISPOOFING_AUTODETECT Antispoofing checks are explicitly turned on on the fingerprint sensor if the device supports it.	1
ABS_ANTISPOOFING_DEVICE_DEFAULT Antispoofing settings are not touched in any way so default settings (device dependent) are used.	2

6.22 Parameter ABS_PARAM_SENSOR_SECURITY Values (ABS_SENSOR_SECURITY_xxxx)

These constants are intended as possible values of parameter

ABS_SENSOR_SECURITY_LOW BSAPI always uses lower security of communication with fingerprint device. It can enhance performance with some fingerprint devices.	0
ABS_SENSOR_SECURITY_NORMAL BSAPI automatically decides what secutritu level to use. How the settings apply then depends on device type and its capabilities. Highly secured communication is usually used whenever possible, unless it would have very bad impact on performance. This is default value.	1
ABS_SENSOR_SECURITY_HIGH BSAPI always uses higher security of communication with fingerprint device. Using this value can have notably negative impact on performance.	2

6.23 Callback Message Codes (ABS_MSG_xxxx)

These codes are used as values for dwMsgID parameter of ABS_CALLBACK.

Callback function can react on various messages accordingly, usually showing/updating a dialog with some message. Also the specific ABS_MSG_xxxx values determine the meaning of the pMsgData parameter of the callback.

There are several categories of the messages:

- Process messages (ABS_MSG_PROCESS_xxxx). These determine lifecycle of the complete interactive operation. Each process is demarcated by ABS_MSG_PROCESS_BEGIN and ABS_MSG_PROCESS_END. Between the two some other messages (including nested subprocess) can arrive. See description of ABS_PROCESS_xxxx constants for more information about the operation lifecycle.
- Prompting messages (ABS_MSG_PROMPT_xxxx). The callback is expected to prompt user to do some action with the FM sensor, e.g. scan his finger, or left the finger from the sensor.
- Quality feedback messages (ABS_MSG_QUALITY_xxxx). These inform use that his interaction with the sensor has low quality. Depending on the nature of the interactive operation, this can lead to repeating the process, so that the user is prompted to do the action again.
- Navigation messages (ABS_NAVIGATE_xxxx). These are called only during navigation (see ABSNavigate).

The following table lists all supported message codes.

ABS_MSG_PROCESS_BEGIN	0x11000000
New process stage of the interactive operation begun. Together with ABS_MSG_PROCESS_END, this messages define the interactive operation lifecycle skeleton. pMsgData points to additional data stored in structure ABS_PROCESS_BEGIN_DATA.	
ABS_MSG_PROCESS_END	0x12000000
Process stage of the interactive operation ended. Together with ABS_MSG_PROCESS_BEGIN, this messages define the interactive operation lifecycle skeleton. See description of ABS_PROCESS_xxxx constants for more information about the operation lifecycle. pMsgData points to additional data stored in structure ABS_PROCESS_DATA.	
ABS_MSG_PROCESS_SUSPEND	0x13000000
Execution of the interactive operation has been suspended. It happens when some other operation (with the same or higher priority) acquires the sensor. After this interrupting operation finishes, the process is resumed again. pMsgData points to additional data stored in structure ABS_PROCESS_DATA.	
ABS_MSG_PROCESS_RESUME	0x14000000
The interactive operation has been resumed. pMsgData points to additional data stored in structure ABS_PROCESS_PROGESS_DATA.	
ABS_MSG_PROCESS_PROGRESS	0x15000000

Informs that the operation is in progress. pMsgData points to additional data stored in structure ABS_PROCESS_DATA.	
ABS_MSG_PROCESS_SUCCESS Informs that the process has succeeded. If it comes it is last message before ABS_MSG_PROCESS_END. Depending on the particular process nature it can use ABS_MSG_PROCESS_SUCCESS or ABS_MSG_PROCESS_FAILURE before ABS_MSG_PROCESS_END, but some other processes do not call any of the two. pMsgData points to additional data stored in structure ABS_PROCESS_SUCCESS_DATA.	0x16000000
ABS_MSG_PROCESS_FAILURE Informs that the operation has failed. If it comes it is last message before ABS_MSG_PROCESS_END. Depending on the particular process nature it can use ABS_MSG_PROCESS_SUCCESS or ABS_MSG_PROCESS_FAILURE before ABS_MSG_PROCESS_END, but some other processes do not call any of the two. pMsgData points to additional data stored in structure ABS_PROCESS_DATA.	0x17000000
ABS_MSG_PROMPT_SCAN Callback should prompt the user to swipe his finger. pMsgData is always NULL.	0x21000000
ABS_MSG_PROMPT_TOUCH Callback should prompt user to touch the sensor. pMsgData is always NULL.	0x22000000
ABS_MSG_PROMPT_KEEP Callback should prompt user to keep the finger on the sensor. pMsgData is always NULL.	0x23000000
ABS_MSG_PROMPT_LIFT Callback should prompt the user to left his finger from the sensor. pMsgData is always NULL.	0x24000000
ABS_MSG_PROMPT_CLEAN Callback should prompt the user to clean the sensor. pMsgData is always NULL.	0x25000000
ABS_MSG_QUALITY Swipe quality is low. Note that if possible BSAPI sends more specific ABS_MSG_QUALITY_xxxx messages, this message is sent only when no more specific message is appropriate.	0x30000000

pMsgData is always NULL.	
ABS_MSG_QUALITY_CENTER_HARDER Swipe quality is low. User should center his finger on the sensor and press harder. pMsgData is always NULL.	0x31000000
ABS_MSG_QUALITY_CENTER Swipe quality is low. User should center his finger on the sensor. pMsgData is always NULL.	0x31100000
ABS_MSG_QUALITY_TOO_LEFT Swipe quality is low. The swipe is too left. pMsgData is always NULL.	0x31110000
ABS_MSG_QUALITY_TOO_RIGHT Swipe quality is low. The swipe is too right. pMsgData is always NULL.	0x31120000
ABS_MSG_QUALITY_TOO_HIGH Swipe quality is low. The swipe is too high. pMsgData is always NULL.	0x31130000
ABS_MSG_QUALITY_TOO_LOW Swipe quality is low. The swipe is too low. pMsgData is always NULL.	0x31140000
ABS_MSG_QUALITY_HARDER User should press harder. pMsgData is always NULL.	0x31200000
ABS_MSG_QUALITY_TOO_LIGHT Swipe quality is low. The swipe is too light. pMsgData is always NULL.	0x31210000
ABS_MSG_QUALITY_TOO_DRY Swipe quality is low. The swipe is too dry. pMsgData is always NULL.	0x31220000
ABS_MSG_QUALITY_TOO_SMALL Swipe quality is low. The swipe is too small. pMsgData is always NULL.	0x31230000
ABS_MSG_QUALITY_TOO_SHORT	0x32000000

Swipe quality is low. The swipe is too short. pMsgData is always NULL.	
ABS_MSG_QUALITY_TOO_FAST Swipe quality is low. The swipe is too fast. pMsgData is always NULL.	0x33000000
ABS_MSG_QUALITY_TOO_SKEWED Swipe quality is low. The swipe is too skewed. pMsgData is always NULL.	0x34000000
ABS_MSG_QUALITY_TOO_DARK Swipe quality is low. The swipe is too dark. pMsgData is always NULL.	0x35000000
ABS_MSG_QUALITY_BACKWARD Swipe quality is low. The swipe is moved backward. pMsgData is always NULL.	0x36000000
ABS_MSG_QUALITY_JOINT Swipe quality is low. Joint has been detected. pMsgData is always NULL.	0x37000000
ABS_MSG_NAVIGATE_CHANGE Notifies about navigation change (user has moved his finger, touched the sensor of left the finger). Applies only during navigation operation. pMsgData points to ABS_NAVIGATION_DATA structure.	0x41000000
ABS_MSG_NAVIGATE_CLICK Notifies that the user clicked on sensor by his finger. Applies only during navigation operation. pMsgData is always NULL.	0x42000000
ABS_MSG_DLG_SHOW Notifies that the feedback dialog should be shown. pMsgData is always NULL.	0x51000000
ABS_MSG_DLG_HIDE Notifies that the feedback dialog should be hidden. pMsgData is always NULL.	0x52000000
ABS_MSG_IDLE Special message, which gives the callback a chance to cancel the interactive operation. pMsgData is always NULL.	0x0

Note that this message is used only when flag ABS_OPERATION_FLAG_USE_IDLE was specified in structure ABS_OPERATION.

This allows to cancel the interactive operation even in single-threaded applications.

7 List of Defined Result Codes

The following result codes can be returned as the PT_STATUS values.

Success return status.	
ABS_STATUS_OK	(0)
General, unknown, or unspecified error.	
ABS_STATUS_GENERAL_ERROR	(-5001)
Internal error.	
ABS_STATUS_INTERNAL_ERROR	(-5002)
BSAPI has been already initialized.	
ABS_STATUS_ALREADY_INITIALIZED	(-5003)
BSAPI is not initialized.	
ABS_STATUS_NOT_INITIALIZED	(-5004)
Connection is already opened.	
ABS_STATUS_ALREADY_OPENED	(-5005)
Invalid parameter.	
ABS_STATUS_INVALID_PARAMETER	(-5006)
Invalid (connection) handle.	
ABS_STATUS_INVALID_HANDLE	(-5007)
No such device found.	
ABS_STATUS_NO_SUCH_DEVICE	(-5008)
Operation has been interrupted due timeout.	
ABS_STATUS_TIMEOUT	(-5009)
Requested feature/function not implemented.	
ABS_STATUS_NOT_IMPLEMENTED	(-5010)
Requested feature/function not supported.	
ABS_STATUS_NOT_SUPPORTED	(-5011)

The operation has been canceled.

ABS_STATUS_CANCELED (-5012)

The operation has not been found (invalid operation ID or the operation already finished).

ABS_STATUS_NO_SUCH_OPERATION (-5013)

Communication error related to remote session (Terminal Services or Citrix)has occured.

ABS_STATUS_REMOTE_COMM_ERROR (-5014)

The operation is not permitted. It might be a matter insufficient rights of the current user.

ABS_STATUS_ACCESS_DENIED (-5015)

There is not enough permanent memory to store the data.

ABS_STATUS_NOT_ENOUGH_PERMANENT_MEMORY (-5016)

8 New Features in Version 4.0

8.1 New Device property

New device proprty ABS_DEVPROP_USAGE was added and can be used with ABSGetDevicePropery.

8.2 New Global Parameters

There are two new global parameters ABS_PARAM_CONSOLIDATION_COUNT_MIN and ABS_PARAM_CONSOLIDATION_COUNT_MAX which replace now obsolete ABS_PARAM_CONSOLIDATION.

8.3 Advanced Support for LEDs

There are new and more versatil functions for controling blinking with LEDs. See description of functions ABSSetLedEx and ABSGetLedEx and related structures, and also the chapter Blinking with LEDs.

9 New Features in Version 3.9

9.1 New Global Parameters

There are two new global parameters ABS_PARAM_POWER_SAVE_ACTIOVE_TO_SLEEP_TIMEOUT and ABS_PARAM_POWER_SAVE_RESET_ON_OPETRATION_START, which allow some detailed settings for power control management.

New global parameter ABS_PARAM_IGNORE_BIOMETRIC_TIMEOUT specifies whether biometric situation is reported as ABS_MSG_QUALITY_TOO_FAST via ABS_CALLBACK.

9.2 New Device Properties.

WIth function ABSGetDeviceProperty two new properties can be inspected: ABS_DEVPROP_SYSTEM_ID and ABS_DEVPROP_SYSTEM_NAME.

10 New Features in Version 3.5

10.1 Global Parameter ABS_PARAM_IFACE_VERSION

New global parameter ABS_PARAM_IFACE_VERSION was added, which allows the caller to detect the interface of the BSAPI because version 3.5 of BSAPI brings one icompatibility with previous version: dynamic enrollment (see below).

The old interface (which does not support this parameter) is version 1. Current version is 2. Future versions of BSAPI will use higher numbers if they will introduce new incompatibilities.

To distinguish between version 1 and 2 of the interface programatically, call ABSGetGlobalParameter() with dwParam set to ABS_PARAM_IFACE_VERSION. If the return value is ABS_STATUS_NOT_SUPPORTED, it's interface version 1. If ABS_STATUS_OK is returned, interpret the returned output parameter to determine number of the interface.

Note that this parameter is designed to be read-only. I.e. you can only get its value with ABSGetGlobalParameter(). Attempting to use it in ABSSetGlboalParameter() will fail with ABS_STATUS_NOT_SUPPORTED.

10.2 Dynamic Enrollment

Since version 3.5 BSAPI.DLL uses a dynamic enrollment. This means that there is no a priori known count how many swipes are required to enroll a finger. During the enrollment process the resulted fingerprint template is actualized and analyzed after each finger scan, and the process finishes when BSAPI evaluates quality of the fingerprint template as sufficient.

This change required several changes in the API.

10.2.1 Global Parameter ABS_PARAM_CONSOLIDATION_COUNT

This global parameter was already present in older version of BSAPI however set of supported values have changed. Old version supported 3 and 5 swipes mostly on all devices. Currently only values 0 (default, meaning the dynamic enrollment) and 5 are supported.

Value 3 is not longer supported since version 3.5.

10.2.2 Structure ABS_PROCESS_BEGIN_DATA

Message ABS_MSG_PROCESS_BEGIN has attached some information, pointed by last parameter sent to ABS_CALLBACK function. The data are described by ABS_PROCESS_BEGIN_DATA.

Now the member StepCount can be set to zero if count of steps is not known, e.g. as in case of dynamic enrollment.

10.2.3 Structure ABS_PROCESS_PROGRESS_DATA

Message ABS_MSG_PROCESS_PROGRESS now comes with more data. Previously it was accompanied with ABS_PROCESS_DATA, now ABS_PROCESS_PROGRESS_DATA is sent.

Note that ABS_PROCESS_PROGRESS_DATA is binary compatible with ABS_PROCESS_DATA. ABS_PROCESS_PROGRESS_DATA is superset of ABS_PROCESS_DATA. Beside the old ProcessID member, ABS_PROCESS_PROGRESS_DATA has member Percentage which informs the caller how the operation progresses.

The process (e.g. the dynamic enrollment) ends when the Percentage reaches 100. For processes where the Percentage is not applicable, it's set to 0xFFFFFFFF.

10.2.4 Constant ABS_PROCESS_CONSOLIDATE

The enrollment process has been changed and the separate finger templates are consolidated into the resulted enrollment template gradually so the consolidation is not single step of the enrollment process.

Member ProcessID in structures ABS_PROCESS_DATA, ABS_PROCESS_BEGIN_DATA, ABS_PROCESS_PROGRESS_DATA and ABS_PROCESS_SUCCESS_DATA is never set to the value. The constant is kept in bstypes.h header for backward compatibility of application source codes which might use it.

10.3 Image Grabbing Functions

10.3.1 Constant ABS_FLAG_HIGH_RESOLUTION

Function ABSGrab accepts new flag ABS_FLAG_HIGH_RESOLUTION, which asks the function to use the highest available image resolution.

10.3.2 Structure ABS_IMAGE

Structure ABS_SAMPLE_IMAGE was renamed to ABS_IMAGE. ABS_SAMPLE_IMAGE is kept as typedefed alias of ABS_IMAGE for backward compatibility.

10.3.3 New Grabbing Functions

Three new brand image grabbing functions are added in BSAPI.DLL version 3.5: ABSListImageFormats, ABSGrabImage and ABSRawGrabImage. All these use new structure ABS_IMAGE_FORMAT to identify supported and desired image formats.

10.4 Global Parameter ABS_PARAM_POWER_SAVE_CHECK_KEYBOARD

New global parameter ABS_PARAM_POWER_SAVE_CHECK_KEYBOARD was added, which allows to tune power management into more details.

10.5 Internal Template Format Types

Please note this article refers only to the data of the template data itself, i.e. the member Data of ABS_BIR structure. All internal template types are always preceded by ABS_BIR_HEADER when returned from BSAPI, and also the ABS_BIR_HEADER preceding the data is expected on input.

In general Upek uses several template format types:

- legacy template,
- alpha template,
- beta template,
- alpha multi-template.

The internal format of the fingerprint teplates returned from the BSAPI.DLL has changed in version 3.5. BSAPI.DLL up to version 3.0 always returned all templates in the legacy format. Since BSAPI.DLL 3.5, enrollment process results in alpha multi-template and verification templates are always beta. Future BSAPI versions might switch to yet another (currently undefined) template format.

BSAPI 3.5 is able to take any of the listed template types on input. When comparing two templates with ABSVerifyMatch function or ABSSrvVerifyMatch, each of the input templates can have different format. So typically you don't need to care from which version the template originates.

If you really need a speicific format of the template, e.g. when you need the template to pass to another library (not part of BSAPI SDK), not supporting too new template types, you may use BCLIB library. BCLIB is now provided as a part of the BSAPI SDK. It provides an interface for converting among various tamplate format types. However remember that some conversions might imply partial data loss of the template, and not all conversions are supported (for example converting legacy template to beta template is not possible). Refer to documentation of the BCLIB for more information about this topic.

10.6 Compatibility with Windows NT Services

Since version 3.5 BSAPI.DLL provides new function ABSInitializeEx, taking a bitmask flags as its only parameter. This function can be (on Windows platform) used to initialize in a mode compatible with Windows NT service.

10.7 ABS_CALLBACK and Threads

Callbacks from interactive operations are now guaranteed to be called from the same thread context, where the interactive operation has been called. This is especially important for developers which use BSAPI.DLL from a programming language without any support for multithreading, e.g. MS VisualBasic.

10.8 Server-Side Library BSSRV.DLL

New library BSSRV.DLL was added into BSAPI SDK. This library offers simple way of comparing two fingerprint templates, without opening any session with fingerprint sensor device.

This library was designed for applications of server-client architecture, where clients are expected to work with fingerprint sensor (i.e. client uses BSAPI.DLL), while server typically manages some database of fingerprint templates and offers verification service for clients.

10.9 Support for Terminal and Citrix

BSAPI now supports usage of locally plugged-in fingerprint devices from remote application, when connected to a remote desktop via Terminal Services or Citrix.

See chapter Support for Terminal Services and Citrix for description of this feature.