Precise iOS Toolkit User Manual

2.13

P/N: ALA 113 1001 - 4160/user RU

1 PRECISE IOS TOOLKIT

1.1 Abbreviations & acronyms

APDU	Application Protocol Data Unit
API	Application Programming Interface
PCSC	Personal Computer/Smart Card
UI	User Interface

1.2 Introduction

This document describes the Precise iOS Toolkit and is intended for the designers and developers of iPhone or iPad applications using the Tactivo^{TM} from Precise Biometrics. It is assumed that the user of this document has knowledge about Objective-C and biometric terms in general.

1.3 Toolkit Architecture

The **Precise iOS Toolkit** consists of the following parts:

- **Precise iOS Library** – the core functionality, i.e. smart card and biometry functionality delivered as a precompiled static library, see chapter 2 and 3 respectively for more information.
- **Precise iOS Reference** – reference implementations of some classes conforming to the `PBBiometryGUI` and `PBBiometryDatabase` protocols as well as additional classes that may be useful when developing applications for the Tactivo reader, see chapter 4 for more information. These classes are delivered in source code.

The folder structure of the delivered toolkit is as follows:

- doc – manuals and reference documentation.
- include – external API header files.
- lib – static library that provides biometric and smart card functionality
- reference – reference implementations that are strongly recommended to be utilized by app developers.

1.4 Sample Code

Sample code and sample applications that show how to use the toolkit can be downloaded from https://premium.precisebiometrics.com/premium-support.

1.5 API Overview

See the corresponding .h-files for details. The PBSmartcard.h, PBBiometry.h, PBAccessory.h and PBLibrary.h are the entry points to the main functionality of the Tactivo reader. Below is the corresponding class diagrams illustrated.

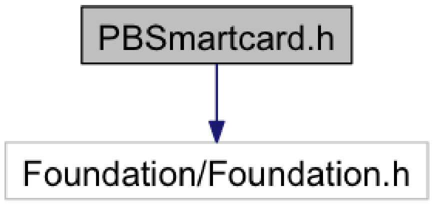


Figure 1 PBSmartcard.h

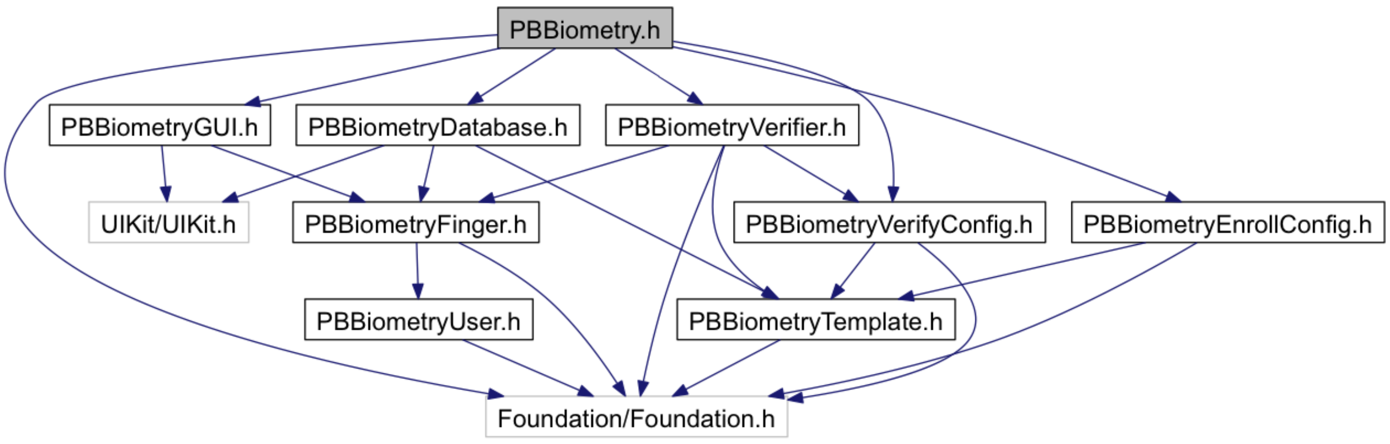


Figure 2 PBBiometry.h

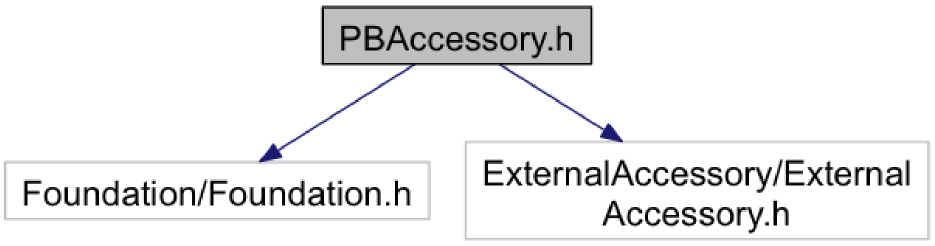


Figure 3 PBAccessory.h

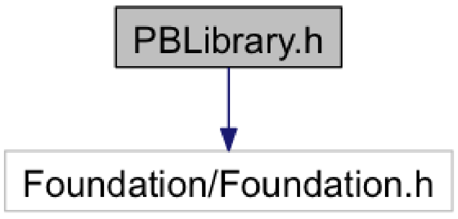


Figure 4 PBLibrary.h

1.6 Using the Toolkit

To be able to make use of the Tactivo reader, the application must include at least the lib and include folders in its project. Precise Biometrics also strongly recommends that the reference folder is included and that the reference implementations are used.

2 SMART CARD

2.1 Introduction

This chapter describes the smart card related parts of the Precise iOS Library and is intended for the developers of iPhone or iPad applications using smart cards with the Tactivo reader from Precise Biometrics.

The purpose of this chapter is to give guidelines for writing new or porting existing code that communicates with smart cards in general. It also addresses some iOS implementation specifics that should be addressed by the developer in order to implement a successful smart card application.

It is assumed that the user of this document has previous knowledge of how an application can communicate with a smart card with APDUs as well as basic knowledge of the ANSI C and Objective-C programming languages.

The implementation notes in this document are valid for iOS 5 and may be valid for future versions as well. Refer to the current iOS documentation for further information.

2.2 Smart card reader API

The Precise iOS Toolkit offers two different APIs to achieve the same thing, to communicate with a smart card from an iOS application. One is a port of the Microsoft de-facto standard ANSI C-implementation of the PCSC workgroup specifications which is also used on Linux and Mac OS X. The other is an objective-C API tailored for the iOS environment.

The PCSC C-API allows for quick and easy porting of existing smart card code. Note that even though all the functions are implemented not all are relevant for an iOS scenario. These functions will return an error code.

The Objective-C API is a proprietary API designed specifically for the iOS environment. This API makes it easy to get started with new projects and for developers not familiar with the full flavor of the PCSC API.

2.3 iOS External Accessory limitations

2.3.1 Background execution

The Tactivo smart card reader is designed according to a set of rules defined by Apple to ensure that the end-user experience will not be negatively affected when introducing an external accessory. Some of these rules affect when and how the accessory can draw power from the internal battery of the iOS device. A simplification of the iOS external accessory power management behavior is that the reader can draw power and communicate with an application when the application is in the foreground and when the screen is active. When the application is moved to the background the application is moved to a suspended state shortly after and when this happens the device cannot assume that there will be enough current available from the iOS device to continue to keep the state of the card. The device will therefore close the session to the card and power off the card.

An application that wants to keep an open session to a card for a prolonged time can ask the system to not be suspended when sent to the background. By doing so the device can be guaranteed power the card for as long as required by the application. How to prevent an application from being suspended is further described in the chapter "Background Execution and Multitasking[1]" in the iOS Developer Library.

When an application is designed to execute in the background the smart card library needs to know not to act upon foreground/background notifications from the system. Remember that it is the responsibility of the application to ensure the user solution performance and the smart card behavior when the automatic power management of the smart card library is disabled. Having an application running in the background with a card powered will affect the battery performance.

2.3.2 Monitoring smart card reader slot events

Another result of the inability to implement system services on the iOS is revealed when attempting to track and monitor slot events in the smart card reader. In order to track the insertion and removal of a smart card in the accessory a session needs to be opened to the accessory via the External Accessory Framework. Once the session is in place the application can monitor the slot events. The library handles the opening and closing of sessions. With the PCSC API the session is initialized and opened when the application calls `SCardEstablishContext()` the first time and closed when the last context is released with `SCardReleaseContext()`.

2.3.3 Shared card sessions

On a PC an application can choose to connect to a smart card in a shared mode meaning that other applications can share the access to the card. This permission is managed by a system service. Due to security concerns Apple has chosen not to allow 3rd party developers to implement a system service for iOS.

The PCSC-API operates in the application's own context and not as a system wide service. This will result in that API-calls and parameters that are aimed for system operations, such as opening sessions to a card in shared mode in order to share the card over several applications is not supported. Attempting to use `SCARD_SHARE_SHARED` together with `SCardConnect()` will return the error `SCARD_E_INVALID_PARAMETER`.

[1]https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html

Opening a session using `SCARD_SHARE_EXCLUSIVE` however guarantees that no other application will be able to access the smart card as long as the session is open.

2.4 External Accessory protocols

2.4.1 Smart card protocol strings

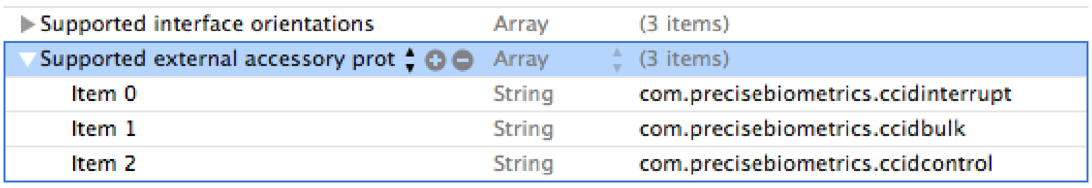
The Tactivo smart card reader communicates with the iOS device over three different protocols. The handling of the protocols is managed by the static library so that the host application does not need to handle the low levels of external accessory management.

The following protocols are used by the library when accessing the smart card functionality in the Tactivo accessory:

- `com.precisebiometrics.ccidinterrupt`
- `com.precisebiometrics.ccidbulk`
- `com.precisebiometrics.ccidcontrol`

These protocol strings must be defined in the application plist file in order to tell the iOS device that the application supports the Tactivo accessory.

To enter a protocol string in an application plist file, add the pre-defined key "Supported external accessory protocols" in the plist file and add the supported protocols.



▶ Supported interface orientations	Array	(3 items)
▼ Supported external accessory prot	Array	(3 items)
Item 0	String	com.precisebiometrics.ccidinterrupt
Item 1	String	com.precisebiometrics.ccidbulk
Item 2	String	com.precisebiometrics.ccidcontrol

Figure 5 – The application plist file in XCode 4

`SCardEstablishContext()` will return `SCARD_E_INVALID_TARGET` if not all protocols are present in the plist file.

Similarly the `open` command in `PBSmartcard` will return `PBSmartcardStatusProtocolNotIncluded` if not all protocols are listed.

2.5 PCSC C-API implementation notes

2.5.1 API limitations

All the standard winscard.h functions are exported in the library but not all functions are fully implemented due to the inherit restraints of the iOS platform. The following functions are exported but will return the error code **`ENOSYS`** indicating that the function is not implemented and will always fail.

`SCardListReaderGroups`

`SCardControl`

The `SCardCancel` will cancel an ongoing call to `SCardGetStatusChange` but it will not cancel an ongoing `SCardTransmit` call.

`SCardBeginTransaction` and `SCardEndTransaction` will always return success as long as the card handle is valid. Transactions are implicitly supported as sharing a connection to a smart card between apps isn't supported on iOS.

2.5.2 Executing code in the background

The vendor specific attribute `SCARD_ATTR_AUTO_BACKGROUND_HANDLING` together with `SCardSetAttrib()` can be used to disable/enable the automatic power management of the smart card when an app is sent to the background. The automatic power management is enabled by default.

The application is responsible to monitor relevant events and request more time from iOS to execute in the background or to make sure to end any ongoing smart card communication and end the session with a call to `SCardReleaseContext()` before iOS forcibly terminates the underlying `EASession` sessions causing undefined behavior in the communication with the library and accessory.

Note that background execution with Tactivo requires at least iOS 5.0. Using the `SCARD_ATTR_AUTO_BACKGROUND_HANDLING` parameter with `SCardSetAttrib()` on earlier versions of iOS will return the error `SCARD_E_READER_UNSUPPORTED.`

2.6 PBSmartCard API implementation notes

2.6.1 Executing code in the background

To manage background events manually via the Objective-C API the method `openWithDisabledBackgroundManagement` is used instead of the regular `open`.

The application is responsible to monitor connect and disconnect events from the `ExternalAccessoryFramework` and either request more time from iOS to execute in the background or make sure to end any ongoing smart card communication and end the session with a call to close before iOS forcibly terminates the underlying `EASession` sessions causing undefined behavior in the communication with the library and accessory.

Note that background execution with Tactivo requires at least iOS 5.0. Calling `openWithDisabledBackgroundManagement` on earlier versions of iOS will return the error `PBSmartcardStatusNotSupported.`

2.6.2 Smart card slot notifications

Once successfully initialized the `PBSmartCard` API will send notifications when the state of the smart card slot changes. An application can choose to listen and act upon these events accordingly.

The following notifications are sent:

`PB_CARD_INSERTED` – is broadcasted when a smart card is inserted in the reader.

`PB_CARD_REMOVED` – is broadcasted when a smart card is removed from the reader.

3 BIOMETRY

This chapter describes the biometry API of the Precise iOS Library and is intended for the designers and developers of iPhone or iPad applications using fingerprint security with the Tactivo from Precise Biometrics.

3.1 Application Architecture

The main biometry functionality is accessed through the `PBBiometry` class. It contains methods for enrolling a finger, `enrollFinger`, and verifying a finger against one or more enrolled fingers, `verifyFingers`.

The `PBBiometry` class also contains a method for simply grabbing images from the sensor, without enrolling or verifying, `captureImagesWithGUI`. This method should mainly be used for creating views in which the user can practice on producing a good fingerprint from the fingerprint sensor, in which case the enrollment and verification functionality is not needed.

The Precise iOS Library is a generalization of the more extended Precise BioMatch Framework, which is a highly modularized framework containing the core biometric functionality. One of the strengths of that framework is that different GUIs can be used while keeping the rest of the application intact. This also applies to the Precise iOS Library, where each application simply has to implement their own class conforming to the `PBBiometryGUI` protocol. The methods of the `PBBiometryGUI` protocol will be called from within the Precise iOS Library, when appropriate. E.g. the `displayImage` method will be called for each image received from the Tactivo.

The same applies for the database in which fingerprint templates (`PBBiometryTemplate`) are stored. Each application may implement its own class conforming to the `PBBiometryDatabase` protocol.

With version 2.0 of the Precise iOS Library an application may also implement its own verifier class conforming to the `PBBiometryVerifier` protocol. This shall mainly be used to support match-on-card, i.e. when the verification shall be done on the smartcard. A new `verifyFingers` method has also been implemented in the `PBBiometry` class to support this. By calling this new method the supplied verifier will be called instead of the default internal verifier. Note that it is not possible to implement your own extractor, hence the new verifier must support one of the supported template types, see `PBBiometryTemplate.h` for more information. To use another template type than the default `PBBiometryTemplateTypeISOCompactCard`, set the `templateType` parameter in the `PBBiometryVerify/EnrollConfig` classes.

Also included in the Precise iOS Toolkit are some reference implementations of classes conforming to the `PBBiometryGUI` and `PBBiometryDatabase` protocols which we strongly recommend that each application uses, see chapter 4 for more information.

The code for communicating with the fingerprint sensor and receiving images from it is hidden within the methods of `PBBiometry`. At the beginning of a call to any of these methods, the fingerprint sensor will be locked by the method and will be released first when the operation is completed. While the fingerprint sensor is locked no other application or thread within the same application will be able to access it, and will instead return `PBBiometryStatusReaderBusy`. If the fingerprint sensor is not attached to the iPhone or the iPad the methods return `PBBiometryStatusReaderNotAvailable`.

3.2 API Overview

See the corresponding .h files for details. Below is a list of the available classes and protocols.

3.2.1 Classes

`PBBiometry`

`PBBiometryTemplate`

`PBBiometryUser`

`PBBiometryFinger`

`PBBiometryEnrollConfig`

`PBBiometryVerifyConfig`

3.2.2 Protocols

`PBBiometryGUI`

`PBBiometryDatabase`

`PBBiometryVerifier`

3.2.3 Raw images

`PBBiometryGUI`'s two methods `displayImage` and `displayChosenImage` are used to display images to the end user. These images have been processed to enhance contrast and remove blur. It can however be useful to have the raw unprocessed image as provided by the sensor, e.g. if using an external extractor. Two methods are provided to facilitate this: `displayRawImage` and `displayChosenRawImage`. See `PBBiometryGUI.h` for more details.

3.3 External Accessory Protocols

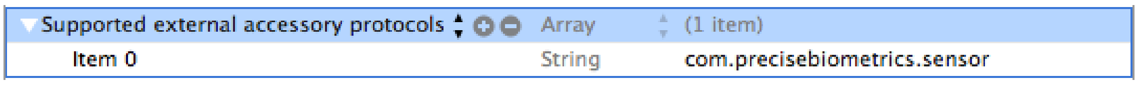
The Tactivo communicates with the iOS device using a single protocol. The communication is managed by the static library so that the host application does not need to handle the low levels of external accessory management.

The following protocol is used by the library when accessing the fingerprint functionality in the Tactivo.

- `com.precisebiometrics.sensor`

This protocol string must be defined in the application plist file in order to tell the iOS device that the application supports the Tactivo.

To enter a protocol string in an application plist file, add the pre-defined key "Supported external accessory protocols" in the plist file and add the supported protocols.



Supported external accessory protocols	Array	(1 item)
Item 0	String	com.precisebiometrics.sensor

Figure 6 – The "Supported external accessory protocols" key in the application plist file.

4 IOS REFERENCE

4.1 Introduction

This chapter describes the reference implementations of the `PBBiometryDatabase` and `PBBiometryGUI` protocols as well as some additional classes and views aimed at applications that want to make use of the Tactivo from Precise Biometrics. Precise Biometrics strongly recommends that these reference implementations are used to keep a consistent UI in regards to biometry between several applications from different vendors. Small changes may be made, e.g. colors of navigation bars and toolbars are easily changeable to fit the general color schemes of the application.

It is assumed that the user of this chapter has knowledge about Objective-C and biometric terms in general.

4.2 Application Architecture

The reference implementations are classes that conform to the `PBBiometryDatabase` and `PBBiometryGUI` protocols as well as a number of other classes. Below is a short summary of the available classes and their functionality. For extended documentation, please refer to the .h files for each class.

Since version 2.0 of the Precise iOS Library includes support for implementing your own verifier, some of the classes in Precise iOS Reference has also been updated to support this feature. The classes `PBEnrollmentController`, `PBVerificationController` and `PBManageFingersController` all include a `PBiometryEnroll/VerifyConfig` parameter to be able to support additional template types. The `PBVerificationController` also include a `id<PBBiometryVerifier>` parameter to support additional verifiers.

4.2.1 PBReferenceDatabase

The `PBReferenceDatabase` class conforms to the `PBBiometryDatabase` protocol and implements a database that stores the biometric templates into a keychain for improved security. See Apple´s "Keychain Services Programming Guide" for more information about Keychain Services.

4.2.2 PBEnrollmentController

The `PBEnrollmentController` class conforms to the `PBBiometryGUI` protocol. It implements a GUI for an enrollment process. Precise Biometrics strongly recommends using this enrollment controller instead of writing your own.

4.2.3 PBVerificationController

The `PBVerificationController` class conforms to the `PBBiometryGUI` protocol. It implements a GUI for a verification process. Precise Biometrics strongly recommends using this verification controller instead of writing your own.

With version 2.0 of Precise iOS Reference it is possible to only verify against one finger, even if several fingers are enrolled. To only verify against one finger, set the `verifyAgainstAllFingers` to `NO`.

4.2.4 PBPracticeController

The `PBPracticeController` class conforms to the `PBBiometryGUI` protocol and implements a GUI for teaching the user how to use the fingerprint sensor. It consists of four steps with the last step allowing the user to practice in order to get as good quality of the fingerprint image as possible. Each step is visualized by the `PBPracticeXController` classes (X = 1-4).

4.2.5 PBManageFingersController

The `PBManageFingersController` class handles finger registration (enrollment) as well as deletion of fingers, for a single user. This controller also makes sure that only enrolled users are allowed to enroll new fingers or re-enroll fingers.

4.2.6 Additional available classes

4.2.6.1 PBDisconnectionViewController

The `PBDisconnectionViewController` class displays an animation of an iPhone being connected to a Tactivo. Use this to display to the user that the Tactivo is not connected to the iPhone.

4.2.6.2 PBFingerImageView

Help class to display a fingerprint image in the different views (e.g. used by `PBEnrollmentController` and `PBPractice4Controller`).

4.2.6.3 PBBarImageView

Help class to display the quality and area bars for the practice view.

4.3 BioMatch3

With version 2.0 of the Precise iOS Reference a couple of classes is included that implements support for match-on-card using the Precise BioMatch3 algorithm. The following classes are included:

4.3.1 PBBioMatch3Database

The `PBBioMatch3Database` class conforms to the `PBBiometryDatabase` protocol for using match-on-card with the Precise BioMatch3 algorithm.

4.3.2 PBBioMatch3Verifier

The `PBBioMatch3Verifier` class conforms to the `PBBiometryVerifier` protocol for using match-on-card with the Precise BioMatch3 algorithm.

4.3.3 PBSmartcard7816

The `PBSmartcard7816` class is a subclass of `PBSmartcard`. Implements the parts of the ISO 7816-4 standard needed by the PBSmartcardBioManager and PBSmartcardIDStore classes.

4.3.4 PBSmartcardBioManager

The `PBSmartcardBioManager` class is a subclass of `PBSmartcard7816`. Implements a high level API for the BioManager card edge.

4.3.5 PBSmartcardIDStore

The `PBSmartcardIDStore` class is a subclass of `PBSmartcardBioManager`. Implements a high level API for the IDStore card edge.

4.4 Sounds and Vibration

In version 2.0 of the Precise iOS Reference we've added the possibility to include sounds and vibration for different events in an application using the Tactivo. It's possible to play a sound when the following events happen:

1. The Tactivo is connected to the device.
2. The Tactivo is disconnected from the device.
3. A smartcard is inserted in the smartcard reader.
4. A smartcard is removed from the smartcard reader.
5. A verification or enrollment fails (due to timeout, cancel or other error).
6. A verification or enrollment succeeds.
7. A swipe fails (due to too poor quality, too small area or a mismatch).

For 7, it is also possible to vibrate the device.

4.4.1 Playing sounds

The triggering of sounds 1-4 is done in Precise iOSLibrary, hence all applications using the library may use these sounds. For sounds 5-7, these are triggered in the reference code (`PBVerificationController` and `PBEnrollmentController`), which means that an application using these controllers (which we recommend) may use these sounds.

The application developer may change which sound that is played for each event, by adding new sounds to the application with the same filename as the default sounds. The sounds shall be named:

1. PBAccessoryConnected.wav
2. PBAccessoryDisconnected.wav
3. PBSmartcardInserted.wav
4. PBSmartcardRemoved.wav
5. PBVerificationRejected.wav
6. PBVerificationAccepted.wav
7. PBSwipeFailed.wav

Besides adding the sound files to the application the application developer also needs to set the value `YES` for the key `PBPlaySounds` in the shared user defaults object:

```
[NSUserDefaults standardUserDefaults] setBool: YES
forKey:@”PBPlaySounds”];
```

If set to `NO` or not set at all, no sounds will play. Applications may add a settings bundle to allow the user to turn on/off sounds.

Note: the device needs to be un-muted for the sounds to play.

4.4.2 Vibrating the device

To vibrate the device when event 7 occurs the application developer needs to set the value `YES` for the key PBVibrateForSwipeFailure in the shared user defaults object:

```
[NSUserDefaults standardUserDefaults] setBool: YES
forKey:@”PBVibrateForSwipeFailure”];
```

If set to `NO` or not set at all, the device will not vibrate. Applications may add options in the settings bundle to allow the user to turn on/off vibration.

4.5 API Overview

See the corresponding .h files for details.

5 MISCELLANEOUS

5.1 Logging

The toolkit has logging functionality that can be enabled at runtime by the consuming app. The logging facility will output messages to Apple System Logger, the XCode console as well as to file.

The log file is stored in the app sandbox at ./Library/Caches/Logs/PreciseBiometricsiOSLibrary.log[2].

The log functionality is managed using the `logLevel` property from PBLibrary.h.

All logging from the toolkit is disabled by default.

Enabling logging adds some computational overhead that may affect overall performance as well as reveal possible sensitive information such as the content of an APDU or fingerprint data. It is therefore recommended to use logging only during app development and for troubleshooting apps in the field.

6 SUPPORT

Please use the Premium Support channel at http://www.precisebiometrics.com/support for best and fastest support.

[2] An additional new file is created with filename PreciseBiometricsiOSLibrary <n>.log (where n is incremented) for each fresh start of the app or if the log file size exceeds 1 MB.