GLOBALPLATFORM®
THE STANDARD FOR SECURE DIGITAL SERVICES AND DEVICES

GlobalPlatform Technology
TEE TUI Extension: Biometrics API
Version 1.0

Public Release

March 2018

Document Reference: GPD_SPE_042

THIS SPECIFICATION OR OTHER WORK PRODUCT IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY IMPLEMENTATION OF THIS SPECIFICATION OR OTHER WORK PRODUCT SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE COMPANY, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER DIRECTLY OR INDIRECTLY ARISING FROM THE IMPLEMENTATION OF THIS SPECIFICATION OR OTHER WORK PRODUCT.

Contents

Tables

Figures

1 Introduction

Many sensitive use cases lead to a user interaction during which the physical presence and acceptance of the authorized user must be confirmed. They are mainly, but not exclusively, related to financial services and corporate usages: bill payment, money transfer, document signature validation, access control to corporate data assets, etc. In the past, verification of the user has often been performed through PIN or password entry.

The TEE Internal Core API ([TEE Core API]) offers the possibility to execute all sensitive operations within a Trusted Application (TA) running in the Trusted Execution Environment (TEE). However, certain applications need to verify the presence of a known user and sometimes to request a conscious gesture to confirm validation or acceptance of information or a transaction. To be trustworthy, these verification operations need to be handled inside the TEE and not to rely on facilities in the Rich Execution Environment (REE); hence the requirement to implement the Biometric Sensor as part of a Trusted User Interface, with its driver amongst the Trusted OS Components. The Biometrics API will be implemented as part of the TEE, and all data will be processed and stored protected by the TEE.

This document defines and specifies:

- The discovery and identification of all biometric capabilities.
- The use of biometric functionality supported by hardware, entirely protected inside the TEE.

This specification is to be used by Trusted Application developers relying on biometric recognition for authentication of the user and confirmation of user acceptance.

The structure of the defined interface is such that alternative biometric functionality can easily be integrated in a logical manner into the function structure. The only functionality that is required to be common to all biometrics is the ability to discover and disclose all biometric capability that may be present in the device. The structures returned are intended to support management of parallel instances of different Biometric Peripherals, and to support the concurrent use of multi-modal biometrics through fusion.

The current set of functions in the API has been validated against the requirements of fingerprint biometrics specifically. We anticipate that the API will be sufficient for a large number of established biometric modalities, but we recognize that some biometric sources may require future extensions. The architecture and structure of this API have been designed to support such future extensions as necessary, and the Discover function provides support for multiple biometric modalities to be available and used, jointly and separately, on one device.

This document is an extension to TEE Trusted User Interface Low-level API ([TEE TUI Low]).

1.1 Audience

This document is intended to support software developers implementing Trusted Applications running inside the TEE which need to verify the user's presence and/or the user's acceptance of a proposed transaction, or implementing Relying Applications (RAs) running in the REE relying on a Relying Trusted Application (RTA) for verification of user presence and/or acceptance.

This document is also intended for TEE implementers who wish to support these biometric interfaces.

1.2 IPR Disclaimer

Attention is drawn to the possibility that some of the elements of this GlobalPlatform specification or other work product may be the subject of intellectual property rights (IPR) held by GlobalPlatform members or others. For additional information regarding any such IPR that have been brought to the attention of GlobalPlatform, please visit https://www.globalplatform.org/specificationsipdisclaimers.asp. GlobalPlatform shall not be held responsible for identifying any or all such IPR, and takes no position concerning the possible existence or the evidence, validity, or scope of any such IPR.

1.3 References

Table 1-1: Normative References

Standard / Specification	Description	Ref
GPD_SPE_010	GlobalPlatform Technology TEE Internal Core API Specification	[TEE Core API]
GPD_SPE_055	GlobalPlatform Technology TEE Trusted User Interface Low-level API	[TEE TUI Low]
ISO/IEC 9899:1999	Programming languages – C	[C99]
RFC 2119	Key words for use in RFCs to Indicate Requirement Levels	[RFC 2119]

Table 1-2: Informative References

Standard / Specification	Description	Ref
GPD_SPE_007	GlobalPlatform Technology TEE Client API Specification	[TEE Client API]
GPD_SPE_009	GlobalPlatform Technology TEE System Architecture	[TEE Sys Arch]
GPD_SPE_021	GlobalPlatform Technology GPD TEE Protection Profile	[TEE Prot Profile]
GPD_SPE_025	GlobalPlatform Technology TEE TA Debug Specification	[TEE Debug]
GPD_SPE_120	GlobalPlatform Technology TEE Management Framework	[TEE Mgmt Fmwk]
GP_GUI_001	GlobalPlatform Document Management Guide	[Doc Mgmt]
ISO/IEC 2382-37:2017	Information Technology – Vocabulary – Part 27: Biometrics	[ISO 2382]
BSI-PP-0084	Security IC Platform Protection Profile with Augmentation Packages – Eurosmart	[BSI-PP-0084]

1.4 Terminology and Definitions

The following meanings apply to SHALL, SHALL NOT, MUST, MUST NOT, SHOULD, and MAY in this document (refer to [RFC 2119]):

- **SHALL** indicates an absolute requirement, as does **MUST**.
- **SHALL NOT** indicates an absolute prohibition, as does **MUST NOT**.
- **SHOULD** indicates a recommendation.
- **MAY** indicates an option.

Our terminology is aligned with ISO/IEC 2382-37/2017 ([ISO 2382]); however, our definitions may vary in precision from those in the ISO document. For the purpose of this document, the definitions in Table 1-3 shall take precedence.

Table 1-3: Terminology and Definitions

Term	Definition
Associate	For an RTA to form a Link with a Stored Template through the TEE, so that subsequently the RTA: • Will consider a successful match against that template as authentication of the Authorized User ID in question. • May also link an action or function with the Stored Template so that a successful match also leads to a call to that action or function. • May also link state change with the Stored Template so that a successful match also enables abilities only available in that state.
Association	The creation (by an RTA) of a Link between a Stored Template and an Authorized User ID. For more details, see *Associate*.
Authorized User	The person expected or required to control the device, the owner, but also the person with access and usage rights to the service or function in the RA. This term should be considered in the context of each RTA separately and multi-user devices have to be taken into account as well as personal devices. The important distinction to make is that the ownership of the RTA and the services and assets it represents is not linked to the ownership of the device and its integrated Biometric Sub-system. The following roles may become an authorized user once validated by the biometric sub-system. o **Biometric (sub-)system owner** (ISO Definition 3.7.9) o **Biometric applicant** (ISO Definition 3.7.1) o **Biometric capture subject** (ISO Definition 3.7.3) o **Biometric data subject** (ISO Definition 3.7.5)
Biometric Peripheral	A component of the TEE and part of the Biometric Sub-system. It represents a selected biometric modality, or set of modalities in multi modal fusion, and software to process it. Multiple Biometric Peripherals may use the same Biometric Sensor and one Biometric Peripheral may use multiple Biometric Sensors at the same time or alternately. It is discovered by TAs through a discovery function provided through the Peripheral API. It is the Biometric Peripheral's task, using whatever capability the Biometric Sensor offers, to provide the functionality required by the Biometrics API.

Term	Definition
Biometric Sensor	Hardware device, including its drivers, physically integrated in the device. A physical component that transforms the user's biometric feature, when presented, into raw or processed biometric data: the Live Image. A Biometric Sensor might be able to provide other functionality such as pre-processing of raw data, extraction of template, or matching.
Biometric Sub-system	The part of the Trusted OS Components specific to implementing the Biometrics API, including supporting hardware, such as Biometric Sensors. In addition to Trusted OS Components, it also includes any untrusted parts such as an REE Enrollment support app.
Biometrics API	The API defined in the current document, TEE TUI Extension: Biometrics API.
Capture	The act by which the Biometric Sensor collects data from the physical biometric feature to create the raw data set, known as the Live Image in the case of biometric traits, which can be visualized as images, e.g. face, iris, or fingerprints. The term **Capture** is equivalent to **Biometric Capture Process** (ISO Definition 3.5.2); in our definition: • *physical biometric feature* is equivalent to **biometric characteristic** (ISO Definition 3.1.2). • *raw data set* is equivalent to **biometric sample** (ISO Definition 3.3.21). • Live Image is equivalent to **biometric sample** (ISO Definition 3.3.14).
Client Application (CA)	An application running outside of the Trusted Execution Environment (TEE) making use of the TEE Client API ([TEE Client API]) to access facilities provided by Trusted Applications inside the TEE. Contrast *Trusted Application (TA)*.
Dissociate	For an RTA to delete the Link it has created between an Authorized User ID known to the RTA and a specific Stored Template, after which the RTA cannot request verification against that Stored Template. Contrast *Associate*.
Dissociation	The deletion (by an RTA) of a Link between a Stored Template and an Authorized User ID. For more details, see *Dissociate*.
End User	Someone who presents a biometric trait to the biometric system and who may or may not have been authenticated. The ISO vocabulary distinguishes between **biometric applicant** (ISO Definition 3.7.1), i.e. an individual seeking to be enrolled, and **claimant** (ISO Definition 3.7.10), i.e. an individual seeking to be verified.
Enroll	To create a new Stored Template for future reference, defining the authorized user with reference to the unique instance of this biometric trait.
Enrollment	The creation of a new Stored Template for future reference, defining the authorized user with reference to the unique instance of this biometric trait. Enrollment is a prerequisite to Association.

Term	Definition

Event API	An API that supports the event loop. Includes the following functions, among others:
	`TEE_Event_AddSources`
	`TEE_Event_OpenQueue`
	`TEE_Event_Wait`
	The Event API is currently defined in [TEE TUI Low], but will be defined in [TEE Core API] beginning with v1.2.

Event loop	A mechanism by which a TA can enquire for and then process messages from types of peripherals including pseudo-peripherals.
Extract (Extraction)	To create a Template from biometric image data (Live Image) through extraction of critical and unique information (the **biometric features** (ISO Definition 3.3.11)) contained in the image and so in the physical biometric trait. The term *Extract* is equivalent to **Biometric Feature Extraction** (ISO Definition 3.5.4).
Fingerprint Sensor	Hardware component which captures an image of the fingerprint when the End User presents the finger to the sensor.
Identification	The Matching of a Live Template against a list of Associated and Stored Templates to identify the best correlation. For more details, see *Identify*.
Identify	To Match a Live Template against a list of Associated and Stored Templates to identify the best correlation. Returns the unique identifier of the best match to the Stored Template, provided a preset minimum of likeness is achieved. In terms of ISO vocabulary, **identify** is defined as a **biometric search** (ISO Definition 3.5.6) against a **biometric enrolment database** (ISO Definition 3.3.9) to find and return the **biometric reference identifier(s)** (ISO Definition 3.3.19).
Link	Internal association between an RTA and a Stored Template. Links are managed by the TEE.
Live Image	Image data captured in real time by the Biometric Sensor as the End User's biometric trait is presented to it. *Live Image* is equivalent to **biometric sample** (ISO Definition 3.3.21).
Live Template	The Template Extracted from the current scan sample, the Live Image. *Live Template* is equivalent to **biometric probe** (ISO Definition 3.3.14).
Match	Any **similarity score** (ISO Definition 37.03.35) higher than the threshold value is considered a match. *Match* is equivalent to the term **biometric comparison** (ISO Definition 3.5.7.)
Metadata	Metadata is other data associated with the template or Biometric Sub-system and used by the Biometric Sub-system for housekeeping. Examples of such metadata may include association links.
Opaque Handle	The TEE uses opaque handles to refer to objects, enabling more implementation options. See [TEE Core API] section 2.4.
Panic	An exception that kills a whole TA instance. See [TEE Core API] section 2.3.3 for full definition.

Term	Definition
Peripheral API	A low-level API that enables a Trusted Application to interact with peripherals via the Trusted OS. Includes the following functions, among others: • `TEE_Peripheral_GetPeripherals` • `TEE_Peripheral_GetStateTable` • `TEE_Peripheral_Open` The Peripheral API is currently defined in [TEE TUI Low], but will be defined in [TEE Core API] beginning with v1.2.
Property	An immutable value identified by a name.
Relying Application (RA)	Any application (TA or CA) that uses the response to a call to Verify to infer the presence and possible acceptance of the Authorized User.
Relying Trusted Application (RTA)	Any TA that uses the response to a call to Verify to guide process decisions. The RTA may specifically communicate information about the Match result (**comparison decision result** (ISO Definition 3.3.26), which can be a **match** (ISO Definition 3.3.31) or **non-match** (ISO Definition 3.3.33)) to a CA using established TEE communication methods. This CA then becomes an RA.
Rich Execution Environment (REE)	An environment that is provided and governed by a Rich OS, potentially in conjunction with other supporting operating systems and hypervisors; it is outside of the TEE. This environment and applications running on it are considered untrusted. Contrast *Trusted Execution Environment (TEE)*.
Rich OS	Typically, an OS providing a much wider variety of features than are provided by the OS running inside the TEE. It is very open in its ability to accept applications. It will have been developed with functionality and performance as key goals, rather than security. Due to its size and needs, the Rich OS will run in an execution environment outside of the TEE hardware (often called an REE – Rich Execution Environment) with much lower physical security boundaries. From the TEE viewpoint, everything in the REE is considered untrusted, though from the Rich OS point of view there may be internal trust structures. Contrast *Trusted OS*.
Secure Element	A tamper-resistant secure hardware component which is used in a device to provide the security, confidentiality, and multiple application environment required to support various business models. May exist in any form factor, such as embedded or integrated SE, SIM/UICC, smart card, smart microSD, etc.
Session	Logically connects multiple functions invoked on a Trusted Application.
Stored Template	A Template created through Enrollment and stored with a unique identifier for use in future Identification and Verification. The term *Stored Template* is equivalent to **biometric reference** (ISO Definition 3.3.16).
Tamper-resistant secure hardware	Hardware designed to isolate and protect embedded software and data by implementing appropriate security measures. The hardware and embedded software meet the requirements of the latest Security IC Platform Protection Profile ([BSI-PP-0084]) including resistance to physical tampering scenarios described in that Protection Profile.

Term	Definition
Template	A condensed vector format data set which captures the essential data of a biometric trait.
Trusted Application (TA)	An application running inside the Trusted Execution Environment (TEE) that provides security-related functionality to Client Applications outside of the TEE or to other Trusted Applications inside the TEE. Contrast *Client Application (CA)*.
Trusted Execution Environment (TEE)	An execution environment that runs alongside but isolated from an REE. A TEE has security capabilities and meets certain security-related requirements: It protects TEE assets from general software attacks, defines rigid safeguards as to data and functions that a program can access, and resists a set of defined threats. There are multiple technologies that can be used to implement a TEE, and the level of security achieved varies accordingly. Contrast *Rich Execution Environment (REE)*.
Trusted OS	An operating system running in the TEE providing the TEE Internal Core API to Trusted Applications. Contrast *Rich OS*.
Trusted Storage	Storage that is protected either by the hardware of the TEE or cryptographically by keys held in the TEE, and that contains data that is accessible only to the Trusted Application that created the data.
Trusted User Interface (Trusted UI or TUI)	A user interface that ensures that the displays and input components are controlled by the TEE and isolated from the REE and even the TAs.
Verification	The Match between a Stored Template and a Live Template. The answer is only Match or No Match. The definition of the term *Verification* corresponds to the definition given for **biometric comparison** (ISO Definition 3.5.7). The **comparison decision result** (ISO Definition 3.3.26) is either a **match** (ISO Definition 3.3.31) or **non-match** (ISO Definition 3.3.33).
Verify	To Match a Stored Template and a Live Template. For more details, see *Verification*.

1.5 Abbreviations and Notations

Table 1-4: Abbreviations and Notations

Abbreviation / Notation	Meaning
API	Application Programming Interface
CA	Client Application
ID	IDentifier
IETF	Internet Engineering Task Force
OS	Operating System
PIN	Personal Identification Number
PNG	Portable Network Graphics
RA	Relying Application
REE	Rich Execution Environment
RFC	Request For Comments; may denote a memorandum published by the IETF
RTA	Relying Trusted Application
SE	Secure Element
TA	Trusted Application
TEE	Trusted Execution Environment
TUI	Trusted User Interface
UI	User Interface

1.6 Revision History

GlobalPlatform technical documents numbered *n*.0 are major releases. Those numbered *n*.1, *n*.2, etc., are minor releases where changes typically introduce supplementary items that do not impact backward compatibility or interoperability of the specifications. Those numbered *n.n*.1, *n.n*.2, etc., are maintenance releases that incorporate errata and precisions; all non-trivial changes are indicated, often with revision marks.

Table 1-5: Revision History

Date	Version	Description
March 2018	1.0	Public Release

2 Biometrics API Objectives

2.1 Target

This specification is targeted at a TEE running within a smartphone or a tablet. A biometrics API can be envisaged for other devices hosting a TEE, but facilities specific to supporting such devices are out of scope of this specification.

Supported smartphones and tablets in this document have at least a Biometric Sensor which SHALL be wired and integral to the device. Remote peripherals are not considered in this specification.

2.2 Purpose

The Biometrics API permits Relying Trusted Applications (RTAs) to verify the authorized user through access to the Biometric Sub-system of the TEE.

2.3 Scope

This specification describes the API, including functions, data, and implementation properties. It makes possible a fully interoperable implementation of biometrics integrated into the TEE. These will be accessible to TAs through an extension of TEE Trusted User Interface Low-level API ([TEE TUI Low]).

3 Overview of Biometric Architecture

Biometric capabilities and their functionality as present in the hardware of the TEE are made available to the TAs. The biometric capabilities are contained in the Biometric Sub-system, consisting of the Biometric Peripherals which use the Biometric Sensors more precisely:

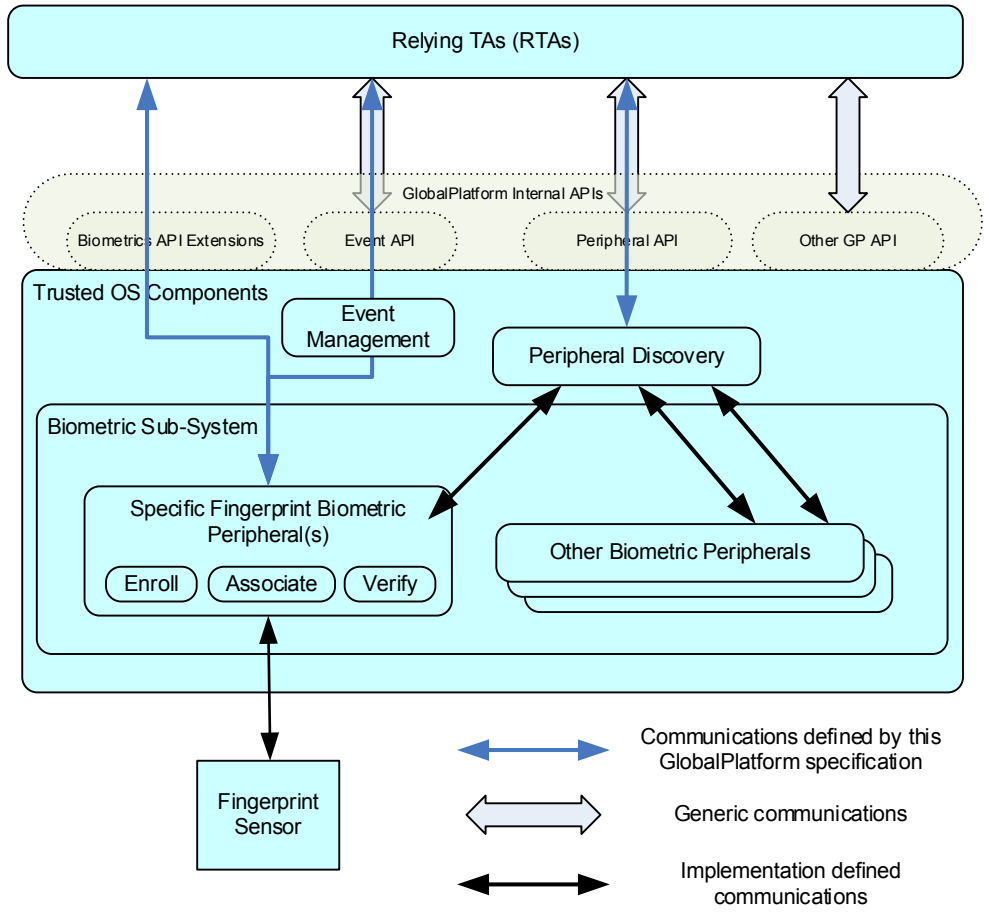
- A Biometric Sub-system is a component of the TEE, composed of all TEE Biometric Peripherals in the device plus any supporting TEE or REE software and hardware.
  - Definition: The part of the Trusted OS Components specific to implementing the Biometrics API, including supporting hardware, such as sensors. In addition to Trusted OS Components, the Biometric Sub-system includes untrusted parts such as an REE Enrollment support app.
  - The Biometric Sub-system exposes the API defined in this document.
  - To discover a Biometric Peripheral, a TA uses the Peripheral API (effectively a different sub-system in the Trusted OS Components) which will communicate in an implementation-specific manner with the Biometric Sub-system to support the discovery of Biometric Peripherals.
- A Biometric Peripheral is a component of the Biometric Sub-system.
  - Definition: A Biometric Peripheral represents a selected biometric modality, or a set of modalities in multi modal fusion, and software to process it. Multiple Biometric Peripherals may use the same sensor and one Biometric Peripheral may use multiple sensors at the same time or alternately. The TAs discover Biometric Peripherals through a discovery function provided through the Peripheral API. It is the Biometric Peripheral's task, using whatever capability the sensor offers, to provide the functionality required by the Biometrics API.
  - The Peripheral API provides an abstracted interface to the biometric sensor which can be used by the TA.
  - The device design and implementation MAY restrict how a Biometric Peripheral can manage access to a sensor or group of sensors, and how the data from that sensor or sensors is exposed to the TA.
  - For data, the Biometric Peripheral only exposes opaque handles to templates and match indications.
  - For access restrictions:
    - The Biometric Peripheral blocks other TAs from using the sensors associated with that Biometric Peripheral when in use.
    - Note that it might achieve this blocking by asking other parts of the Trusted OS Components to manage the blocking.
  - The Biometric Peripheral manages access control to that peripheral's templates.
    - Note that it might achieve this access control by asking other parts of the Biometric Sub-system components to manage the Biometric Peripheral's Templates.
- A Biometric Sensor provides the Live Image and possibly other related services.
  - Definition: A physical component, transforming the user's biometric feature, when presented, into raw or processed biometric data; the Live Image. A Biometric Sensor might be able to provide other functionality such as pre-processing of raw data, extraction of template or matching.
- In general, it is an implementation choice as to whether particular functionality is implemented in the generic Biometric Sub-system, the Biometric Peripheral, or the Biometric Sensor. Such decisions SHALL be transparent to the calling TA.

When interacting with the Biometric Sub-system of the TEE, the first step is the discovery of the available biometric capabilities present in the platform. This is performed using the standard discovery mechanisms in the Peripheral API. The functions used are `TEE_Peripheral_GetPeripherals` and `TEE_Peripheral_GetStateTable`. These functions are defined in [TEE TUI Low] but will be defined in [TEE Core API] beginning with v1.2.

The Biometric Peripheral appears as a peripheral with the type `TEE_PERIPHERAL_BIO`, while the Biometric Sensor will appear with the specific type; for example, `TEE_PERIPHERAL_FINGERPRINT`. The TA calling will only interact with the Biometric Peripheral (`TEE_PERIPHERAL_BIO`); it will not interact directly with the Biometric Sensor.

Once the biometric capabilities are known, the TA can select which Biometric Peripheral to engage with and proceed to utilize that service, as shown for the specific case of fingerprint biometrics in Figure 3-1.

Figure 3-1: Architecture Overview – Multiple Biometrics

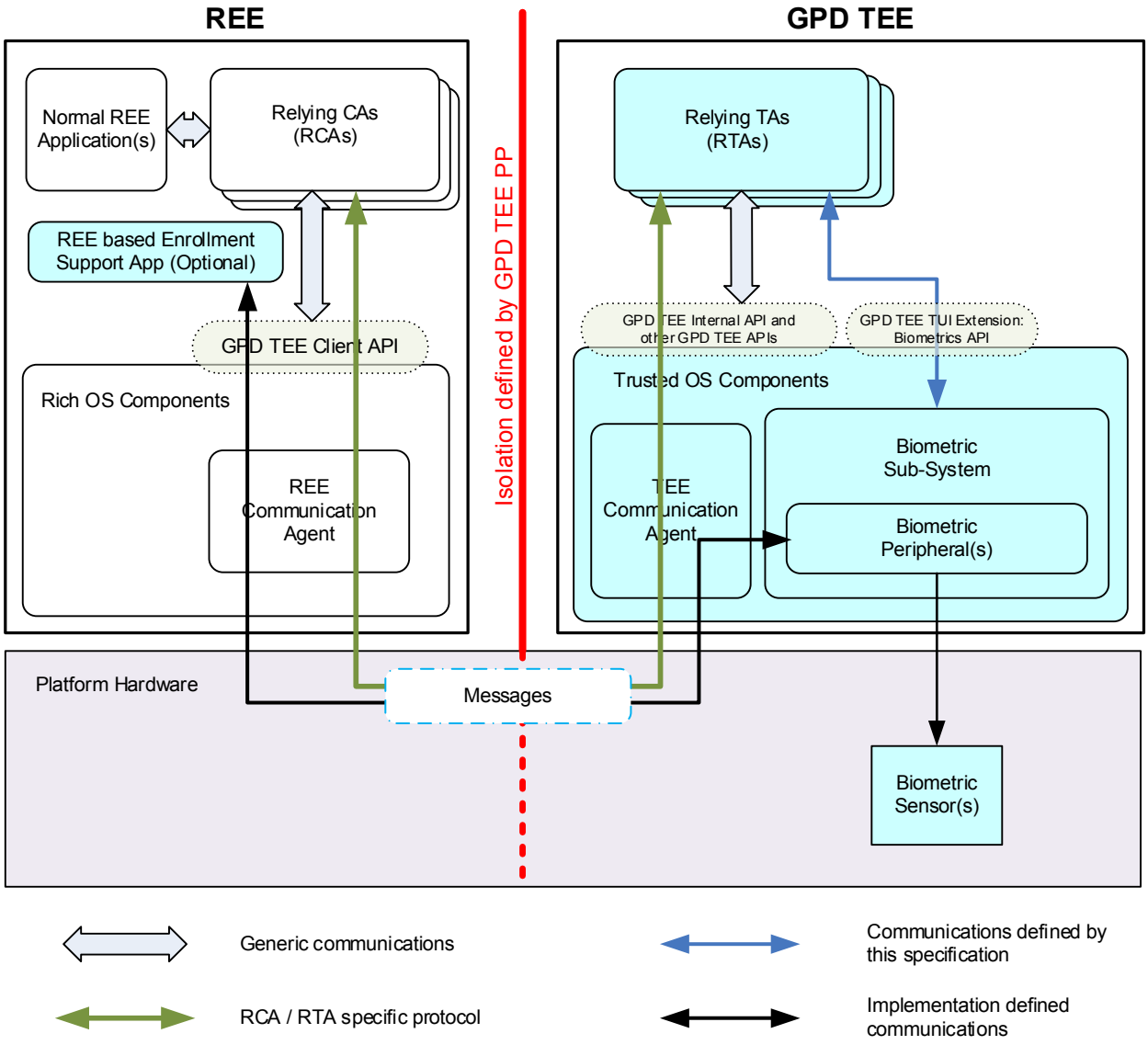


The Biometric Sub-system is integrated into the TEE and provides a service through the established interfaces. It will utilize TEE secure storage, along with REE and SE capabilities as appropriate and available on any specific platform. Figure 3-2 shows the general positioning of the Biometric Sub-system, the Biometric Peripheral, and the Biometric Sensor in a conceptual TEE architecture. Note that part or all of the Biometric Peripheral may optionally be implemented as TAs executing on the TEE, or in one of the available SEs executing as “Match on Card”. In addition, some functionality that is not security-critical may, in some implementations, be handled by Biometric Sub-system components in the REE. Each positioning will provide different advantages and limitations; the choice of architecture in this respect is left to the device manufacturer. However, regardless of where the Biometric Sub-system is placed, its execution and all data, whether long term stored or run-time, SHALL be protected using the security criteria of the TEE for Trusted Storage. Generally, this means that material held on the TEE’s behalf must be bound to the TEE and cannot be accessed or manipulated by unauthorized applications in the TEE, REE, or SE. The Biometric Sub-system described in this specification is, regardless of where it is placed, considered part of the Trusted OS for the purposes of security evaluation.

A Relying TA communicates only with the Biometric Sub-system; this communication API is defined in this document. The Biometric Sub-system communicates with the Biometric Peripheral, which may communicate with a Biometric Sensor, to complete any demanded biometric activity.

The interfaces between the Biometric Sub-system, Biometric Peripheral, and Biometric Sensor are internal to the specific implementation and are out of scope of this document.

Figure 3-2: Architecture Overview – Biometrics



A TEE design may provide the UI experience for the biometric enrollment operation internally or via REE software that is indicated here as the optional Enrollment Support App. The Enrollment Support App is an optional REE piece of software that may be used by the TEE (in conjunction with the TEE-based enrollment sub-system) to provide the UI experience to users who are undergoing the complex enrollment process. How the Enrollment Support App is connected to the TEE, and the relevant parts of the Biometric Sub-system, are implementation-specific.

3.1 API Function Calls Overview

The following functionality is required from the Biometric Sub-system:

- **Enroll:** Creates one new Stored Template from Live Images captured in the operation.
- **Capture:** Initiates the process of capturing biometric data.
- **Verify:** Performs the match between designated Stored Templates and the Live Template.
- **Associate:** Associates one Stored Template with an RTA.
- **Dissociate:** Releases the connection between an RTA and a specific Stored Template. The opposite of Associate.
- **List Templates**: Lists currently Stored Templates.
- **Stop:** Stops a process.

3.1.1 Enroll

Enroll tells the Biometric Sub-system to create a Stored Template and store it securely with an associated unique identifier.

1. Capture one or more images of the End User's biometric feature and create a Template.
2. Optionally, provide user guidance in the process.
3. If able to create a Template of sufficient quality:
   a. Store the created Template as a Stored Template.
   b. Create and retain one unique identifier for the new Stored Template.
4. If unable to create a Template of sufficient quality, no Template is created and an error is returned.

3.1.2 Capture

Capture tells the Biometric Sub-system to start capturing biometric data.

1. Initiate event to prepare the Biometric Peripheral to accept input.
2. Indicate when user first presents their biometric feature.
3. When capture is complete, provide an opaque handle to the captured data, now considered a Live Template.

3.1.3 Verify

Verify tells the Biometric Sub-system to match a Live Template captured in a previous Capture operation against one or more Stored Templates. This makes it possible to confirm the identity of the End User is as claimed, or to identify one End User's biometric feature from a list of Stored Templates. In ISO terms, it comprises **biometric comparison** (the match and decision) and serves both for **biometric identification** (when performed against a list of Stored Templates) and **biometric verification** (when performed against only one Stored Template, which represents the claim).

The ISO vocabulary defines **biometric verification** (ISO Definition 3.8.3) as the process of confirming a **biometric claim** (ISO Definition 3.6.4) through **biometric comparison** (ISO Definition 3.5.7).

1. Perform Match between one Live Template and a list of Stored Templates. Alternatives are:
   a. The list is one index to one Stored Template only; that is the Verify is attempted against only one claimed identity.
   b. The list contains indexes to several Stored Templates that the RTA has previously associated with.

3.1.4 Associate

Associate tells the Biometric Sub-system to increase the count of RTAs associated with a Stored Template. It supports housekeeping functions in the Biometric Sub-system.

1. Match all available Stored Templates against a Live Template provided by the End User.
2. If a Match is found, then:
   a. Create a Link between the calling RTA and one specific Stored Template. Provide one unique identifier to the Stored Template which will be used by the RTA to invoke that Stored Template for Verification or Dissociation.
3. If no Match is found, an error is returned.

3.1.5 Dissociate

Dissociate tells the Biometric Sub-system to decrease the count of RTAs associated with a Stored Template. It supports housekeeping functions in the Biometric Sub-system.

1. Break link between RTA and a specific Stored Template.
2. May provoke an automatic erase of the template.

3.1.6 List Templates

List Templates tells the Biometric Sub-system to return a list of the Stored Templates currently associated with the calling TA.

3.1.7 Stop

Stop tells the Biometric Sub-system to stop the Associate, Capture, Enroll, or Verify operation.

3.2 Data Object Lifecycle

To implement this API, the Biometric Sub-system SHALL manage Live Templates, Stored Templates, and Associations.

3.2.1 Live Templates

A Live Template is the data captured by the sensor. A Live Template is created by a Capture operation started when the TA calls the `TEE_BioStartCapture` function.

The template data SHALL only be available to the Biometric Peripheral.

The Biometric Peripheral SHALL store the Live Template in storage that is at least as secure as TEE working memory.

Each Biometric Peripheral stores a single Live Template. When a Capture operation starts, any stored Live Template SHALL be deleted.

The Biometric Peripheral SHALL ensure that the data presented as a Live Template was captured by the current Capture operation. It SHALL not be possible to re-present data captured in an earlier operation. Therefore, the Live Template SHOULD be stored in volatile memory. If the Live Template is stored in non-volatile memory, the Biometric Peripheral SHALL ensure that it is deleted or invalidated when the TA closes the Biometric Peripheral or starts a new Capture operation.

If the operation is successful, the Biometric Sub-system returns an opaque handle to the Live Template to the calling TA in the `TEE_EVENT_BIO_CAPTURE_COMPLETED` event. As with other opaque handles, this handle is only valid in the context of the TA instance that created it.

If a TA instance attempts to use an opaque handle issued to a different TA instance, that handle SHALL be invalid and the TEE SHALL panic.

While the handle to a template SHALL remain valid for as long as a TA instance is open, the TA SHOULD use a Live Template without delay. If there is a delay between capture and verification or association, the data associated with the handle may no longer exist and the Biometric Sub-system will return an error.

3.2.2 Stored Templates

A stored template is the template created when a user enrolls and is used to verify or associate a Live Template. A stored template is created by an Enroll operation started when the TA calls the `TEE_BioStartEnroll` function.

Stored templates belong to the Biometric Sub-system and SHALL NOT be accessible to any TA or other sub-system or API.

A TA cannot access the stored template; it only knows the `TEE_TemplateID` through its association with that template.

Stored templates SHALL be stored in non-volatile storage that is at least as secure as TEE trusted storage.

Stored templates SHOULD persist over a power cycle and over upgrades to the TEE or upgrades to the Biometric Sub-systems.

This API does not provide a mechanism to install templates captured on another device. A Biometric Peripheral MAY support a proprietary mechanism for loading templates and creating associations with them.

Stored templates SHALL be deleted:

- when the last association with the template is removed
- on factory reset

3.2.3 Associations

An association indicates that a given TA has associated a user's biometric feature with a given stored template. An Association is created by an enroll or an Associate operation started when the TA calls the `TEE_BioStartEnroll` or `TEE_BioStartAssociate` function.

The Biometric Sub-system SHALL store a table of which templates are associated with which TAs, so that it can return a `TEE_TemplateID` whenever a TA verifies biometric data. This table SHALL be stored in storage that is only accessible to the Biometric Sub-system. This storage SHALL be at least as secure as the TEE Trusted Storage.

For any TA instance, the Biometric Sub-system SHALL return the same `TEE_TemplateID` every time a Live Template is verified against a given stored template. The Biometric Sub-system MAY return the same `TEE_TemplateID` for a given stored template to every TA with which it is associated, or it MAY return different `TEE_TemplateIDs` to different TAs.

There is no security advantage to returning different IDs. The only information a TA learns on a successful association is that the user has previously enrolled this biometric data with another TA on the system – but not which TA. The value of the ID does not add any information.

Note: The Biometric Sub-system merely records the fact that specific biometric data has been enrolled and associated with this TA. It does NOT store any data about the identity of the user to whom the template belongs or the purpose for which they want to use that biometric feature. For example, in a banking application a user may choose to associate their index finger with their checking account and the ring finger with a savings account.

The TA SHALL store this information itself. It may be that in their email application the index finger is associated with the user's business email. However, the banking TA cannot know this.

The Biometric Sub-system should maintain all associations in non-volatile memory. Associations SHALL persist over a power cycle and over upgrades to the TEE, Biometric Sub-systems, or TAs.

The Biometric Sub-system SHALL delete an association when:

- A TA calls the Dissociate function.
- The TA that created the association is uninstalled.

4 Biometrics API

4.1 Standard TEE Terminology and Methodology

TEE TUI Extension: Biometrics API is an extension of [TEE TUI Low], and as such the generic definitions from that document apply here.

In the following cases, this specification enhances or modifies those definitions.

4.1.1 Parameter Annotations

In the descriptions of functions, parameters are annotated in accordance with [TEE Core API] section 3.4.

4.1.1.1 Inbufs and Outbufs

This specification makes use of `[inbuf]` and `[outbuf]` (as described in [TEE Core API] sections 3.4.3 and 3.4.4), but instead of using a `void*` pointer for the buffer, it uses a typed pointer. However, the size parameter always refers to a number of bytes.

4.1.2 Data Types

This specification uses the data types defined in [TEE Core API] section 3.2, plus those listed below.

Note: These additional data types will be defined in [TEE Core API] v1.2 and later.

- `size_t`: The unsigned integer type of the result of the sizeof operator.
- `uint64_t`: Unsigned 64-bit integer

In the event of any difference between the definitions in this specification and those in the C99 standard (ISO/IEC 9899:1999 – [C99]), C99 shall prevail.

4.1.3 Return Codes Including Error Codes Used in this Specification

Synchronous functions return `TEE_RESULT` values which are `uint32_t`. Functions that start asynchronous operations also return an immediate result but return further information in events which are added to an event queue. These events are listed in section 4.3.5.

In the case of an error, there is an error event which includes a `TEE_Result` value to indicate the error.

Standard return codes are defined in [TEE Core API]. Additional return codes are listed in section 4.3.1.

4.1.4 External Functions

This specification refers to the following functions defined in other GlobalPlatform specifications:

Table 4-1: External Functions

Function	Specification
TEE_Event_AddSources TEE_Event_OpenQueue TEE_Event_Wait	Event API introduced in [TEE TUI Low] These functions will be defined in [TEE Core API] beginning with v1.2.
TEE_Panic	[TEE Core API]
TEE_Peripheral_GetPeripherals TEE_Peripheral_GetStateTable TEE_Peripheral_Open	Peripheral API introduced in [TEE TUI Low] These functions will be defined in [TEE Core API] beginning with v1.2.
TEE_TUI_BlitDisplaySurface TEE_TUI_GetDisplayInformation TEE_TUI_GetDisplaySurface TEE_TUI_GetPNGInformation TEE_TUI_GetSurface TEE_TUI_GetSurfaceBuffer TEE_TUI_GetSurfaceInformation TEE_TUI_InitSessionLow TEE_TUI_SetPNG	[TEE TUI Low]

4.1.5 Specification Version Number Property

This specification defines a TEE property containing the version number of the specification that the implementation conforms to. The property can be retrieved using the normal Property Access Functions defined in [TEE Core API]. The property SHALL be named "`gpd.tee.biometric.version`" and SHALL be of integer type with the interpretation given below.

The specification version number property consists of four positions: major, minor, maintenance, and RFU. These four bytes are combined into a 32-bit unsigned integer as follows:

- The major version number of the specification is placed in the most significant byte.
- The minor version number of the specification is placed in the second most significant byte.
- The maintenance version number of the specification is placed in the second least significant byte. If the version is not a Maintenance version, this SHALL be zero).
- The least significant byte is reserved for future use. Currently this byte SHALL be zero.

Table 4-2: Specification Version Number Property – 32-bit Integer Structure

Bits [24-31] (MSB)	Bits [16-23]	Bits [8-15]	Bits [0-7] (LSB)
Major version number of the specification	Minor version number of the specification	Maintenance version number of the specification	Reserved for future use. Currently SHALL be zero.

So, for example:

- Specification version 1.1 will be held as `0x01010000` (16842752 in base 10).
- Specification version 1.2 will be held as `0x01020000` (16908288 in base 10).
- Specification version 1.2.3 will be held as `0x01020300` (16909056 in base 10).
- Specification version 12.13.14 will be held as `0x0C0D0E00` (202182144 in base 10).
- Specification version 212.213.214 will be held as `0xD4D5D600` (3570783744 in base 10).

This places the following requirement on the version numbering:

- No specification can have a Major or Minor or Maintenance version number greater than 255.

4.1.6 Header File

The header file for the Biometrics API SHALL have the name "`tee_tui_bio_api.h`".

```
#include "tee_tui_bio_api.h"
```

4.1.7 API Version

The header file SHALL contain version specific definitions from which TA compilation options can be selected.

```
#define TEE_BIO_API_MAJOR_VERSION ([Major version number])
#define TEE_BIO_API_MINOR_VERSION ([Minor version number])
#define TEE_BIO_API_MAINTENANCE_VERSION ([Maintenance version number])
#define TEE_BIO_API_VERSION (TEE_BIO_API_MAJOR_VERSION << 24) +
                            (TEE_BIO_API_MINOR_VERSION << 16) +
                        (TEE_BIO_API_MAINTENANCE_VERSION << 8)
```

The document version-numbering format is ***X.Y[.z]***, where:

- Major Version (X) is a positive integer identifying the major release.
- Minor Version (Y) is a positive integer identifying the minor release.
- The optional Maintenance Version (z) is a positive integer identifying the maintenance release.

`TEE_BIO_API_MAJOR_VERSION` indicates the major version number of the Biometrics API. It SHALL be set to the major version number of this specification.

`TEE_BIO_API_MINOR_VERSION` indicates the minor version number of the Biometrics API. It SHALL be set to the minor version number of this specification. If the minor version is zero, then one zero shall be present.

`TEE_BIO_API_MAINTENANCE_VERSION` indicates the maintenance version number of the Biometrics API. It SHALL be set to the maintenance version number of this specification. If the maintenance version is zero, then one zero shall be present.

The definitions of “Major Version”, “Minor Version”, and “Maintenance Version” in the version number of this specification are determined as defined in the GlobalPlatform Document Management Guide ([Doc Mgmt]). In particular, the value of `TEE_BIO_API_MAINTENANCE_VERSION` SHALL be zero if it is not already defined as part of the version number of this document. The “Draft Revision” number SHALL NOT be provided as an API version indication.

A compound value SHALL also be defined. If the Maintenance version number is `0`, the compound value SHALL be defined as:

```
#define TEE_BIO_API_[Major version number]_[Minor version number]
```

If the Maintenance version number is not zero, the compound value SHALL be defined as:

```
#define TEE_BIO_API_[Major version number]_[Minor version number]_[Maintenance
version number]
```

Some examples of version definitions:

For TEE TUI Extension: Biometrics API v1.3, these would be:

```
#define TEE_BIO_API_MAJOR_VERSION        (1)
#define TEE_BIO_API_MINOR_VERSION        (3)
#define TEE_BIO_API_MAINTENANCE_VERSION  (0)
#define TEE_BIO_API_1_3
```

And the value of `TEE_BIO_API_VERSION` would be 0x01030000.

For a maintenance release of the specification as v2.14.7, these would be:

```
#define TEE_BIO_API_MAJOR_VERSION        (2)
#define TEE_BIO_API_MINOR_VERSION        (14)
#define TEE_BIO_API_MAINTENANCE_VERSION  (7)
#define TEE_BIO_API_2_14_7
```

And the value of `TEE_BIO_API_VERSION` would be 0x020E0700.

4.1.8 Version Compatibility Definitions

A TA can set the definitions in this section to non-zero values if it was written in a way that requires strict compatibility with a specific version of this specification. These definitions could, for example, be set in the TA source code, or they could be set by the build system provided by the Trusted OS, based on metadata that is out of scope of this specification.

This mechanism can be used where a TA depends for correct operation on the older definition. TA authors are warned that older versions are updated to clarify intended behavior rather than to change it, and there may be inconsistent behavior between different Trusted OS platforms where these definitions are used.

This mechanism resolves all necessary version information when a TA is compiled to run on a given Trusted OS.

```
#define TEE_BIO_REQUIRED_MAJOR_VERSION          (major)
#define TEE_BIO_REQUIRED_MINOR_VERSION          (minor)
#define TEE_BIO_REQUIRED_MAINTENANCE_VERSION    (maintenance)
```

The following rules govern the use of `TEE_BIO_API_REQUIRED_MAJOR_VERSION`, `TEE_BIO_REQUIRED_MINOR_VERSION`, and `TEE_BIO_REQUIRED_MAINTENANCE_VERSION` by TA implementers:

- If `TEE_BIO_REQUIRED_MAINTENANCE_VERSION` is defined by a TA, then `TEE_BIO_REQUIRED_MAJOR_VERSION` and `TEE_BIO_REQUIRED_MINOR_VERSION` SHALL also be defined by the TA.
- If `TEE_BIO_REQUIRED_MINOR_VERSION` is defined by a TA, then `TEE_BIO_REQUIRED_MAJOR_VERSION` SHALL also be defined by the TA.

If the TA violates any of the rules above, TA compilation SHALL stop with an error indicating the reason.

`TEE_BIO_REQUIRED_MAJOR_VERSION` is used by a TA to indicate that it requires strict compatibility with a specific major version of this specification in order to operate correctly. If this value is set to `0` or is unset, it indicates that the latest major version of this specification SHALL be used.

`TEE_BIO_REQUIRED_MINOR_VERSION` is used by a TA to indicate that it requires strict compatibility with a specific minor version of this specification in order to operate correctly. If this value is unset, it indicates that the latest minor version of this specification associated with the determined `TEE_BIO_REQUIRED_MAJOR_VERSION` SHALL be used.

`TEE_BIO_REQUIRED_MAINTENANCE_VERSION` is used by a TA to indicate that it requires strict compatibility with a specific major version of this specification in order to operate correctly. If this value is unset, it indicates that the latest maintenance version of this specification associated with `TEE_BIO_REQUIRED_MAJOR_VERSION` and `TEE_BIO_REQUIRED_MINOR_VERSION` SHALL be used.

If **none** of the definitions above is set, a Trusted OS SHALL select the most recent version of this specification that it supports.

If the Trusted OS is unable to provide an implementation matching the request from the TA, compilation of the TA against that Trusted OS SHALL fail with an error indicating that the Trusted OS is incompatible with the TA. This ensures that TAs originally developed against previous versions of this specification can be compiled with identical behavior, or will fail to compile.

If the above definitions are set, a Trusted OS SHALL behave exactly according to the definitions for the indicated version of the specification, with only the definitions in that version of the specification being exported to a TA. In particular an implementation SHALL NOT enable APIs which were first defined in a later version of this specification than the version requested by the TA.

If the above definitions are set to `0` or are not set, then the Trusted OS SHALL behave according to this version of the specification.

As an example, consider a TA which requires strict compatibility with TEE TUI Extension: Biometrics API v1.0:

```
#define TEE_BIO_REQUIRED_MAJOR_VERSION          (1)
#define TEE_BIO_REQUIRED_MINOR_VERSION          (0)
#define TEE_BIO_REQUIRED_MAINTENANCE_VERSION    (0)
```

Due to the semantics of the C preprocessor, the above definitions SHALL be defined before the main body of definitions in "`tee_tui_bio_api.h`" is processed. The mechanism by which this occurs is out of scope of this document.

4.1.9 Structure Versions

It is expected that TAs will be loaded onto devices over the air using the TEE Management Framework ([TEE Mgmt Fmwk]), so it will be useful to be able to write a TA to work with different versions of the API. Therefore, each structure defined in this specification that the implementation can return has a version field and a union of the different versions. As this is the first release, all structures currently only have one member in this union. Each function that returns a structure enables the TA to specify the version of the structure it wants.

An implementation of this API should be able to issue the current version of a structure and such older versions as are defined by that version of the specification and any configuration documents.

A TA should check the version of the API at start up.

If the TEE implements an earlier version, the TA should determine whether it can work with the versions of structures defined in that version of the specification. If it cannot, it should exit gracefully.

If the TA requests an old version, the API should return this version if it can or else return an error if the version is no longer supported.

If the TA requests a version that is higher than that supported by the API, the API should return the most recent version it supports.

The TA should always check the version of the structure returned before attempting to use it.

Table 4-3: Structure Versions

Structure	Version in API 1.0
`TEE_Event_Bio`	1

4.2 Background Material and Implementation Options

4.2.1 Biometric Peripherals

This API is designed to work with many different Biometric Peripherals.

The Biometric Sub-system will consist of a Biometric Sensor and functionality that can match the output from the Biometric Sensor to Stored Templates.

Some systems, such as those for fingerprint recognition, have a dedicated Biometric Sensor. Others, such as facial recognition, use a general-purpose sensor, in this case a camera, to capture the data.

The Biometric Peripheral and the Biometric Sensor appear as separate peripheral types.

When the TA calls a biometric function, the Biometric Sub-system will attempt to lock the required peripherals, and will return an error if it cannot. The TA only needs to lock the Biometric Peripheral.

4.2.2 Live Images, Templates, and Raw Images

Depending on the implementation, the Biometric Sensor may convert the image into a template or it may return a handle to the raw image. Under no circumstances must the TA be able to access the image or template, so this handle may only be used to access the image or template by the underlying TEE.

- When attempting to create a template, the Biometric Sensor will be able to determine whether it has enough information to create a template.
- When returning a handle to the raw data, the Biometric Sensor may not be able to provide immediate feedback as to whether the captured image is a valid biometric sample; this information is only provided once the image is processed, using the `TEE_BioStartAssociate` or the `TEE_BioStartVerify` functions. In either case, at some point if the image does not contain enough data, an error is returned and the RTA will need to prompt the user to re-present their biometric feature.

This information is returned as the `TEE_BIO_CAPTURE_TYPE` field in the peripheral state table.

In either case, the data from the Biometric Sensor must not be available outside the Biometric Sub-system. This data must therefore be stored in a TEE memory area not accessible to general TAs, in TEE Trusted Storage, or in a location considered more isolated and immutable than the TEE but which is bound to the TEE cryptographically (e.g. in a Secure Element file system encrypted against TEE held keys). All references to this data are passed using an opaque handle of type `TEE_BioHandle`.

Any errors processing the image into a template may be returned immediately or as events. Errors where the user may need prompting are returned as events. It is the duty of the TA to update the display to guide the user.

4.2.3 Using Biometrics with the TUI Low-level API

The Event and Peripheral APIs are defined in [TEE TUI Low], but will be defined in [TEE Core API] beginning with v1.2.

When using the Event and Peripheral APIs:

- The TA SHALL use the function `TEE_Peripheral_GetPeripherals` to get a list of all peripherals.
- Peripherals of type `TEE_PERIPHERAL_BIO` are Biometric Peripherals.
- The TA MAY examine the peripheral state table to determine which Biometric Peripheral to use.
- The TA SHALL open the required peripherals.
- The TA MAY open other event sources such as a touch display to allow users to signal inability to present a biometric feature or a desire to use an alternative authentication method such as a PIN.
- The TA SHALL use the function `TEE_Event_AddSources` to add any required `TEE_EventSourceHandle` to the `TEE_EventQueue` used by the TA.
- The TA SHALL call the appropriate biometric function `TEE_BioStartCapture` (section 4.6.2.2) or `TEE_BioStartEnroll` (section 4.6.2.3).
- The TA SHALL monitor the events returned and take the appropriate action. This may involve updating the display to prompt the user.
- After a capture, the TA SHALL call `TEE_BioStartAssociate` (section 4.6.2.1) or `TEE_BioStartVerify` (section 4.6.2.4).
- When finished using the Biometric Sub-system, the TA SHOULD remove the `TEE_EventSourceHandle` from the `TEE_EventQueue` and close the Biometric Peripheral and any other resources opened by the TA for this task.

4.2.4 Biometric Matching Thresholds

Because there is currently no standard way to describe thresholds, this version of the API does not provide information on the threshold values used by the Biometric Sub-system and provides no means for the TA to directly set the threshold being applied. It is assumed that all Biometric Sensors and Biometric Peripherals will use a default value selected by the manufacturer.

This specification does support two assurance levels; it is an implementation detail for the Biometric Peripheral to choose how to implement the different levels. For example, the Biometric Peripheral MAY use a lower matching threshold or MAY switch off liveness detection or other features, when operating at the lower assurance level.

4.2.5 Use of Trusted Storage

The Biometric Sub-system uses Trusted Storage (or an equally secure equivalent) to maintain templates and metadata. If it uses a system outside of the TEE, such as Match on Card, then that system must provide the same characteristics and at least the same security strength as the Trusted Storage system. Therefore, any biometric function may report one of the standard storage errors (see [TEE Core API] Table 3-3: API Return Codes).

For example, while storage systems should attempt to maintain the integrity of the files, they cannot guarantee that data will not be corrupted and therefore must report such corruption.

If the Biometric Sub-system discovers that a template has become corrupted, that template will automatically be deleted and it will no longer be possible to associate or verify against it. If the stored metadata becomes corrupted, access to all templates will be lost.

In case of factory reset:

- Template storage lists SHALL be treated as if they were held in `TEE_STORAGE_PRIVATE`.
- Biometric sessions are terminated.

4.3 Data Constants

4.3.1 Return Codes

In addition to the return codes specified in [TEE Core API], new ones are used in this specification.

For more information on these return codes, including the agreed interpretation valid for that function context, see the function descriptions in section 4.6, where all used return codes are listed.

Return codes associated with the Biometrics API SHALL be of the form 0x*abbbccdd* where:

Table 4-4: Return Code Structure

Digit	Value	Meaning
a	0x0	For all Warnings
	0xf	For all Errors
bbb	0x042	SHALL be set to the document number in binary coded decimal: therefore 042 for codes associated with the Biometrics API
cc	0x00	Errors or warnings that apply to all peripheral types.
	0x01-ff	SHALL be set to the value of the bottom 8 bits of TEE_BIO_TYPE – so 01 for fingerprint errors.
dd	0x00-7f	Values defined by GlobalPlatform Specifications
	0x80-ff	Implementation defined values

Table 4-5: Return Codes

Return Code	Value
Reserved for warnings that apply to all peripheral types	0x00420000 – 0x0042007f
Implementation defined warnings that apply to all peripheral types	0x00420080 – 0x004200ff
TEE_WARN_FP_CAPTURE_TOO_FAST	0x00420100
TEE_WARN_FP_CAPTURE_TOO_SLOW	0x00420101
TEE_WARN_FP_CAPTURE_TOO_FAR_LEFT	0x00420102
TEE_WARN_FP_CAPTURE_TOO_FAR_RIGHT	0x00420103
TEE_WARN_FP_CAPTURE_TOO_FAR_UP	0x00420104
TEE_WARN_FP_CAPTURE_TOO_FAR_DOWN	0x00420105
Reserved for future fingerprint warnings	0x00420106 – 0x0042017f
Implementation defined fingerprint warnings	0x00420180 – 0x004201ff
Reserved for future peripheral warnings	0x0042xx00 – 0x0042xx7f where xx is the bottom 8 bits of a TEE_BIO_TYPE yet to be defined
Implementation defined peripheral warnings	0x0042xx80 – 0x0042xxff where xx is the bottom 8 bits of a TEE_BIO_TYPE yet to be defined

Return Code	Value
TEE_ERROR_NO_MATCH	0xf0420000
TEE_ERROR_INSUFFICIENT_DATA	0xf0420001
TEE_ERROR_SENSOR_ERROR	0xf0420002
TEE_ERROR_PERIPHERAL_IN_USE	0xf0420003
TEE_ERROR_MISSING_LIVE_TEMPLATE	0xf0420004
TEE_ERROR_INSUFFICIENT_ASSURANCE	0xf0420005
Reserved for future errors that apply to all peripherals	0xf0420006 – 0xf042007f
Implementation defined errors that apply to all peripherals	0xf0420080 – 0xf04200ff
Reserved for future fingerprint errors	0xf0420100 – 0xf042017f
Implementation defined fingerprint errors	0xf0420180 – 0xf04201ff
Reserved for future peripheral errors	0xf042xx00 – 0xf042xx7f where xx is the bottom 8 bits of a TEE_BIO_TYPE yet to be defined
Implementation defined peripheral errors	0xf042xx80 – 0xf042xxff where xx is the bottom 8 bits of a TEE_BIO_TYPE yet to be defined
Values not defined above may be defined in other GlobalPlatform TEE specifications.	All other values

4.3.2 TEE_BIO_ASSURANCE_LEVEL

`TEE_BIO_ASSURANCE_LEVEL` is a `uint32_t` used to enumerate the assurance level at which a capture and subsequent verification is to be performed.

An Enroll operation is always performed at assurance level High and an Associate operation may only be performed with a capture made at assurance level High. However, once an association has been made, a TA may choose to capture biometric data at assurance level Low – in which case the subsequent Verify operation is also made at this level.

For compatibility, all Biometric Peripherals must offer both assurance levels – but they MAY choose to offer the same assurance at both levels.

At assurance level Low, a Biometric Peripheral may perform fewer liveness checks, use fewer invariants, or in the case of a fusion biometric, use fewer biometric modalities.

There is no mechanism to compare the assurance between different biometric modalities. The TA developer must take its own steps to ensure that the actual assurance offered by a Biometric Peripheral corresponds to the assurance level suitable for their needs.

Table 4-6: TEE_BIO_ASSURANCE_LEVEL Values

Constant Name	Value	Description
`TEE_BIO_ASSURANCE_HIGH`	`0x00001000`	Higher or equal assurance than TEE_BIO_ASSURANCE_LOW
`TEE_BIO_ASSURANCE_LOW`	`0x00000100`	Lower or equal assurance than TEE_BIO_ASSURANCE_HIGH
Reserved	`0x00000000 – 0x7ffffffe`	Except as defined above.
`TEE_BIO_ASSURANCE_ILLEGAL_VALUE`	`0x7fffffff`	This value is reserved for testing and validation. It SHALL NOT be generated or used in normal operation.
Implementation defined	`0x80000000 – 0xffffffff`	

4.3.3 TEE_BIO_CAPTURE_TYPE

`TEE_BIO_CAPTURE_TYPE` is a `uint32_t` used to enumerate the different ways a Biometric Sensor can return data for processing. It is returned in the peripheral state table (see Table 4-10).

Table 4-7: TEE_BIO_CAPTURE_TYPE Values

Constant Name	Value	Description
`TEE_BIO_CAPTURE_TEMPLATE`	`0x00000001`	Biometric Sensor converts raw data into a template. The TA will receive a handle to that template.
`TEE_BIO_CAPTURE_RAW`	`0x00000002`	The Biometric Sensor returns raw data. The TA will receive a handle to that raw data.
Reserved	`0x00000000 – 0x7ffffffe`	Except as defined above.
`TEE_BIO_CAPTURE_ILLEGAL_VALUE`	`0x7fffffff`	This value is reserved for testing and validation. It SHALL NOT be generated or used in normal operation.
Implementation defined	`0x8000000 – 0xffffffff`	

4.3.4 TEE_BIO_TYPE

`TEE_BIO_TYPE` is a `uint32_t` used to enumerate the different biometric modalities. It is returned in the peripheral state table (see Table 4-10).

Table 4-8: TEE_BIO_TYPE Values

Constant Name	Value	Description
`TEE_BIO_FINGERPRINT`	`0x00000001`	Biometric modality is fingerprint.
Reserved	`0x00000000 – 0x7ffffffe`	Except as defined above.
`TEE_BIO_ILLEGAL_VALUE`	`0x7fffffff`	This value is reserved for testing and validation. It SHALL NOT be generated or used in normal operation.
Implementation defined	`0x80000000 – 0xffffffff`	

4.3.5 TEE_EVENT_BIO_TYPE

TEE_EVENT_BIO_TYPE is a uint32_t used to enumerate the different event types generated by the Biometrics API.

For more information, see the function descriptions in section 4.6.

Table 4-9: TEE_EVENT_BIO_TYPE Values

Constant Name	Value	Data Field	Type
TEE_EVENT_BIO_CAPTURE_READY	0x00000000	TEE_HANDLE_NULL	TEE_BioHandle
TEE_EVENT_BIO_CAPTURE_STARTED	0x00000001	TEE_HANDLE_NULL	TEE_BioHandle
TEE_EVENT_BIO_CAPTURE_FAILED	0x00000002	TEE_HANDLE_NULL	TEE_BioHandle
TEE_EVENT_BIO_CAPTURE_ INSUFFICIENT_DATA	0x00000003	TEE_HANDLE_NULL	TEE_BioHandle
TEE_EVENT_BIO_CAPTURE_COMPLETED	0x00000004	Opaque handle	TEE_BioHandle
TEE_EVENT_BIO_ASSOCIATE_STARTED	0x00000005	TEE_HANDLE_NULL	TEE_BioHandle
TEE_EVENT_BIO_ASSOCIATE_COMPLETED	0x00000006	TemplateID	TEE_TemplateID
TEE_EVENT_BIO_ASSOCIATE_NO_MATCH	0x00000007	TEE_HANDLE_NULL	TEE_BioHandle
TEE_EVENT_BIO_ENROLL_READY	0x00000008	TEE_HANDLE_NULL	TEE_BioHandle
TEE_EVENT_BIO_ENROLL_STARTED	0x00000009	TEE_HANDLE_NULL	TEE_BioHandle
TEE_EVENT_BIO_ENROLL_COMPLETED	0x0000000a	TemplateID	TEE_TemplateID
TEE_EVENT_BIO_VERIFY_STARTED	0x0000000b	TEE_HANDLE_NULL	TEE_BioHandle
TEE_EVENT_BIO_VERIFY_MATCH	0x0000000c	TemplateID	TEE_TemplateID
TEE_EVENT_BIO_VERIFY_NO_MATCH	0x0000000d	TEE_HANDLE_NULL	TEE_BioHandle
TEE_EVENT_BIO_STOPPED	0x0000000e	TEE_HANDLE_NULL	TEE_BioHandle
TEE_EVENT_BIO_ERROR	0x0000000f	Error code	TEE_Result
Reserved for Future Use	0x00000010-0x7ffffffe		
TEE_EVENT_BIO_ILLEGAL_VALUE	0x7fffffff	This value is reserved for testing and validation. It SHALL NOT be generated or used in normal operation.	
Implementation defined general events	0x80000000-0xffffffff		

4.4 State Table

The peripheral state table for a Biometric Peripheral SHALL contain the values listed in Table 4-10.

Table 4-10: TEE_PERIPHERAL_BIO State Table Values

TEE_PeripheralValueType.id	TEE_PeripheralValueType.u
TEE_PERIPHERAL_STATE_BIO_CAPTURE_TYPE	TEE_BIO_CAPTURE_TYPE
TEE_PERIPHERAL_STATE_BIO_TYPE	TEE_BIO_TYPE

4.5 Data Structures

This section defines data structures specific to this specification. In addition, this specification makes use of types and parameter hints (e.g. [in]) defined in [TEE Core API].

4.5.1 TEE_BioHandle

TEE_BioHandle is an opaque handle on the data returned from TEE_BioStartCapture in events with type TEE_EVENT_TYPE_BIO, which the TA can then pass to TEE_BioStartAssociate or TEE_BioStartVerify:

```
typedef struct __TEE_BioHandle* TEE_BioHandle;
```

The handle can only be used in biometric functions. Attempting to access the data by using a biometric handle with the Trusted Storage API functions (described in [TEE Core API]) will cause a panic.

Depending on the internals of the Biometric Sub-system, the data may be a raw image or it may be a template.

4.5.2 TEE_BioTag

```
typedef TEE_BioTag uint32_t;
```

As Biometric Peripherals MAY support multiple concurrent operations, each operation is associated with a tag parameter.

The tag does not need to be unpredictable; it MAY be a simple counter. It is permissible for different Biometric Peripherals to use the same tag, but for any TA instance the combination of Biometric Peripheral and tag SHALL uniquely identify an operation. The tag is generated and supplied by the relevant event generator function, TEE_BioStart*xxxxx*, described below.

4.5.3 TEE_Event_Bio

The TEE_BioStartAssociate, TEE_BioStartCapture, TEE_BioStartEnroll, and TEE_BioStartVerify functions generate events with type TEE_EVENT_TYPE_BIO. The TA SHALL use the TEE_Event_Wait function to collect the events. The payload field of these events is a TEE_Event_Bio structure generated by the Biometric Peripheral and this event source must have been added to the calling TA event queue, as per section 4.6.2.

```
typedef struct {
    uint32_t                version;
    union{
        TEE_Event_Bio_V1    v1;
        } u;
} TEE_Event_Bio;

typedef struct {
    uint32_t                tag;
    TEE_EVENT_BIO_TYPE      bioType;
    TEE_Event_Bio_Return    data;
} TEE_Event_Bio_V1;
```

The return type is one element of the following union, as defined by the value of TEE_EVENT_BIO_TYPE (see Table 4-9).

```
typedef union
        {
        TEE_Result          error;
        TEE_TemplateID      templateId;
        TEE_BioHandle       handle;
    } TEE_Event_Bio_Return;
```

These events are produced by user interaction with a Biometric Sensor.

`version` The version of the included union.

- On entry, the highest version the TA understands.
- On return, the actual version returned – this will always be lower or equal the requested version.

As this is the first release, there is only one supported version.

For more information, see section 4.1.9, Structure Versions.

`tag` The tag parameter of the function that started the process that generated the event.

`bioType` The type of event; for example, capture started. See Table 4-9.

`error` A TEE_Result value encoding an error (TEE_EVENT_BIO_ERROR events).

`templateId` The TEE_TemplateID of the Template to which the event refers, or “ZERO” if no TEE_TemplateID is returned

(See events:

TEE_EVENT_BIO_ASSOCIATE_COMPLETED

TEE_EVENT_BIO_ENROLL_COMPLETED

TEE_EVENT_BIO_VERIFY_MATCH)

`handle`	An opaque handle to live capture data in raw or template forms (`TEE_EVENT_BIO_CAPTURE_COMPLETED` events).

4.5.4 TEE_TemplateID

A `uint32_t` used as a TEE allocated reference number to a Stored Template.

```
typedef uint32_t TEE_TemplateID;
```

"ZERO" is an invalid Template ID which is used to indicate that no Stored Template is available, or could be identified. All other values are assignable by the Biometric Sub-system to identify a specific Stored Template.

The Template ID SHALL be unique for each Stored Template on a per Biometric Peripheral basis. Different Biometric Peripherals may use the same Template ID value.

4.6 Biometric Functions

Sequence diagrams for the biometric functions are found in Annex C.

The specification number and function numbers listed are used by the TEE TA Debug Specification ([TEE Debug]) to provide feedback to developers when debug is active.

4.6.1 Functions That Do Not Require a TUI Session

These functions may be called outside of a TUI session.

4.6.1.1 TEE_BioDissociateTemplate

```
TEE_Result TEE_BioDissociateTemplate(
[in]   TEE_EventSourceHandle      bioPeripheral,
[in]   TEE_TemplateID             templateId
);
```

Description

The `TEE_BioDissociateTemplate` non-TUI function removes the association with the biometric template referenced by the id, and places no requirement on a TUI session. After dissociation, the template can no longer be used by the calling RTA as part of verification. The system MAY at this point choose to garbage collect the template data and it MAY be impossible to re-associate with this template, so the enrollment process would have to be re-executed.

Specification Number: 42 Function Number: 0x0201

Parameters

`bioPeripheral`	A handle referencing the target Biometric Peripheral.
`templateId`	The Template ID to be dissociated.

Return Value

`TEE_SUCCESS`	In case of success.
`TEE_ERROR_NO_DATA`	If there was no association for `templateId`.
`TEE_ERROR_CORRUPT_OBJECT`	If an object the Biometric Sub-system needed to access was corrupt.
`TEE_ERROR_STORAGE_NOT_AVAILABLE`	If the storage system in which an object the Biometric Sub-system needs is not accessible for some reason.
`TEE_ERROR_ACCESS_CONFLICT`	If while opening an object the Biometric Sub-system needs, an access right conflict was detected.
`TEE_ERROR_ITEM_NOT_FOUND`	If for any object the Biometric Sub-system needs to access, the storage does not exist or the object cannot be found in the storage.
`TEE_ERROR_STORAGE_NO_SPACE`	If insufficient space is available to create a new persistent object.

Panic Reasons

- If `bioPeripheral` does not refer to a Biometric Peripheral.
- If the Implementation detects any error associated with this function which is not explicitly associated with a defined return code for this function.

4.6.1.2 TEE_BioListTemplates

```
TEE_Result TEE_BioListTemplates(
[in]       TEE_EventSourceHandle   bioPeripheral,
[outbuf]   TEE_TemplateID*         templateIds, size_t* size
);
```

Description

The `TEE_BioListTemplates` non-TUI function lists the `templateIds` associated with the current TA. This function places no requirement on a TUI session.

Specification Number: 42 Function Number: 0x0202

Parameters

`bioPeripheral`	A handle referencing the target Biometric Peripheral.
`templateIds`	A pointer to an array of `TEE_TemplateID`. The buffer parameter of the `[outbuf]`.
`size`	The `size` field of an `[outbuf]`.

Return Value

`TEE_SUCCESS`	In case of success.
`TEE_ERROR_SHORT_BUFFER`	If the provided buffer is too small. See `[outbuf]` handling in [TEE Core API].
`TEE_ERROR_CORRUPT_OBJECT`	If an object the Biometric Sub-system needed to access was corrupt.
`TEE_ERROR_STORAGE_NOT_AVAILABLE`	If the storage system in which an object the Biometric Sub-system needs is not accessible for some reason.
`TEE_ERROR_ACCESS_CONFLICT`	If while opening an object the Biometric Sub-system needs, an access right conflict was detected.
`TEE_ERROR_ITEM_NOT_FOUND`	If for any object the Biometric Sub-system needs to access, the storage does not exist or the object cannot be found in the storage.
`TEE_ERROR_STORAGE_NO_SPACE`	If insufficient space is available to create a new persistent object.

Panic Reasons

- If `size` is `NULL`.
- See [TEE Core API] section 3.4.4 for reasons for `[outbuf]` generated panic.
- If the Implementation detects any error associated with this function which is not explicitly associated with a defined return code for this function.

4.6.2 Functions That Require a Low-level TUI Session

These functions are to be used with the Event and Peripheral APIs, which are currently defined in [TEE TUI Low] but will be defined in [TEE Core API] beginning with v1.2.

The TA must:

1. Before calling one of these functions first obtain a handle for the Biometric Peripheral using the `TEE_Peripheral_GetPeripherals` and `TEE_Peripheral_Open` functions.
2. Either use the `TEE_Event_OpenQueue` function to open a `TEE_EventQueue` to receive events from the Biometric Peripheral, or use the `TEE_Event_AddSources` function to add the Biometric Peripheral to an existing open queue.
3. Tell the Biometric Peripheral which operation to perform using `TEE_BioStartEnroll` or `TEE_BioStartCapture` followed by `TEE_BioStartVerify` or `TEE_BioStartAssociate`, and retrieve the results as events using the `TEE_Event_Wait` function.
4. Once the required operations are complete, either use the `TEE_Event_DropSources` function to remove the Biometric Peripheral from the event queue or close the event queue.
5. Close the Biometric Peripheral.

The functions described in this section invoke asynchronous operations. It is possible to have multiple operations invoked by these functions running at the same time. The number of simultaneous operations is implementation dependent and SHALL be at least 1. If you attempt to start a function and too many operations are active, the function will return `TEE_ERROR_BUSY`.

As some Biometric Peripherals may be able to deal with multiple operations, each operation is associated with a tag parameter; see section 4.6.2.1.

The TA may process events from other sources while dealing with events from the Biometric Sub-system. For example, the TA may provide the user the option of entering a PIN using a secure display or presenting their biometric feature. In this case, it would need to ensure the event source from the touch display is on the same event queue as the Biometric Peripheral.

The implementation MAY deliver additional events related to the acquisition of biometric data prior to delivering the result.

When using these biometric functions, except for `TEE_BioStartEnroll`, it is the duty of the TA to tell the user what actions to take by displaying information using the TUI Low-level functions.

If several Biometric Peripherals are present and it is not clear which one the user will use, the TA can call `TEE_BioStartCapture` or `TEE_BioStartEnroll` on each in turn, then wait for the events on all those Peripherals. Identify which one returns the `TEE_EVENT_BIO_CAPTURE_STARTED` event, then stop the other Peripherals using the `TEE_BioStop` function.

4.6.2.1 TEE_BioStartAssociate

```
TEE_Result TEE_BioStartAssociate(
[in]   TEE_EventSourceHandle      bioPeripheral,
[in]   TEE_BioHandle              data,
[out]  TEE_BioTag*                tag
);
```

Description

The `TEE_BioStartAssociate` function starts the process of associating the input returned by `TEE_BioStartCapture` with this TA using the Biometric Peripheral referenced by `bioPeripheral`.

To make an association, the biometric data SHALL be captured at assurance level High.

The biometric input SHALL be matched against all existing enrolled templates. If the biometric input matches a template that the calling TA has not already associated with, a new association SHALL be created.

The function can be used to perform a Match against any and all Stored Templates Enrolled in the system. As specified, `TEE_EVENT_BIO_ASSOCIATE_COMPLETED` will signal a successful Match. If a TA wishes to know that the live template matches a stored template that resides on the device, does not care why the stored template was enrolled on the device, and has no interest in storing and managing a relationship between itself and the stored template, then upon receiving `TEE_EVENT_BIO_ASSOCIATE_COMPLETED` the TA should immediately invoke `TEE_BioDissociateTemplate` to prevent Stored Template to TA binding from blocking the garbage collection of the stored template by the biometric sub-system.

Note that the calling TA will have no control over what Stored Templates are Enrolled in the system by other software and will have to make assumptions about the level of authorization that can be attributed to an enrolled template's presence. We therefore recommend that many use cases use the template ID returned by the successful Match, plus a back channel such as password validation, to provide initial binding of the template to owner identity.

The operation is cancellable with the `TEE_BioStop` function.

If the operation is terminated by an error, an error status will be returned through the event queue. For a full list of events that may be returned, see section 4.3.5.

The function is designed to return promptly and report further progress on the invoked operation through events. The function's return does not mean that the biometric data could be associated with a template.

While the operation is active, the Biometric Sub-system will intercept events from the peripherals needed by the Biometric Sub-system.

Refer to the relevant sequence diagram in Annex C for details.

Expected Events for Successful Association

`TEE_EVENT_BIO_ASSOCIATE_STARTED`	Sent to confirm that this Biometric Peripheral is about to perform an Associate operation; information only.
`TEE_EVENT_BIO_ASSOCIATE_COMPLETED`	Sent when association has completed or an existing association match is found; contains the `templateId`.

Failure Events

TEE_EVENT_BIO_ASSOCIATE_NO_MATCH	Sent if association failed because the Live Image could not be matched against any of the existing templates. If the RTA wants to retry the association at this point, it will need to start over and capture new biometric input from the user.
TEE_EVENT_BIO_ERROR	An error or warning has occurred.
TEE_EVENT_BIO_STOPPED	The operation was stopped using the TEE_BioStop function.

Specification Number: 42 Function Number: 0x0301

Parameters

bioPeripheral	The handle to the event source associated with the target Biometric Peripheral.
data	A pointer to an opaque handle to biometric data.
tag	A pointer to the tag used to identify this operation, in events or in the TEE_BioStop function. See section 4.6.2.1.

Return Value

TEE_SUCCESS	The function completes and the operation reports further progress via events.
TEE_ERROR_INSUFFICIENT_ASSURANCE	If the capture was not performed at a high enough assurance level to make an association.
TEE_ERROR_MISSING_LIVE_TEMPLATE	If the Live Template referenced by data is no longer available.
TEE_ERROR_NO_DATA	If there are no enrolled templates. The operation will not be started.
TEE_ERROR_OUT_OF_MEMORY	If the system ran out of resources. The operation will not be started.
TEE_ERROR_BUSY	If the TUI resources are currently in use. The operation will not be started.

Panic Reasons

- If bioPeripheral does not refer to a Biometric Peripheral.
- If tag pointer is NULL.
- If the biometric sub-system fails to load any critical code component.
- If the Implementation detects any error associated with this function which is not explicitly associated with a defined return code for this function.

4.6.2.2 TEE_BioStartCapture

```
TEE_Result TEE_BioStartCapture(
[in]   TEE_EventSourceHandle      bioPeripheral,
       TEE_BIO_ASSURANCE_LEVEL    level
[out]  TEE_BioTag*                tag
);
```

Description

The `TEE_BioStartCapture` function tells the Biometric Peripheral to start capturing biometric data, which may then be associated with or verified against a Stored Template, using biometric input from the Biometric Peripheral referenced by `bioPeripheral`. This will free any current Live Templates before starting the capture.

The operation is cancellable with the `TEE_BioStop` function. If the operation is cancelled, an event will be returned to signal cancellation and no further events will be generated by this operation.

If the Biometric Peripheral encounters a problem which it deems to be recoverable, a warning code is returned through the event queue, see section 4.3.1. The calling TA is expected to handle the warning event through appropriate information to the End User.

If the operation is terminated by an error, an error status will be returned through the event queue. For a full list of events that may be returned, see section 4.3.1.

The function is designed to return promptly and report further progress on the invoked operation through events. The prompt return does not indicate that the user presented their biometric feature or that the biometric data could be captured.

While the operation is active, the Biometric Sub-system will intercept events from the peripherals needed by the Biometric Sub-system.

Expected Events

`TEE_EVENT_BIO_CAPTURE_READY`	Sent when the Biometric Peripheral is ready to accept input. Do not prompt the user for input until this event has been received. If user presents their biometric feature before READY is received, this will be ignored.
`TEE_EVENT_BIO_CAPTURE_STARTED`	Sent when user first presents their biometric feature; information only. Calling TA will always receive both a START and COMPLETED event.
`TEE_EVENT_BIO_CAPTURE_COMPLETED`	Sent when capture is complete; contains an opaque handle to the captured data. This handle can be used in a call to `TEE_BioStartAssociate` or `TEE_BioStartVerify`. Calling TA will always receive both a START and COMPLETED event.

Failure Events

TEE_EVENT_BIO_CAPTURE_FAILED	Sent if the sensor failed to capture an image.
TEE_EVENT_BIO_CAPTURE_INSUFFICIENT_DATA	Sent if the image could not be converted into a template as it did not contain enough information.
TEE_EVENT_BIO_ERROR	An error or warning has occurred.
TEE_EVENT_BIO_STOPPED	The operation was stopped using the TEE_BioStop function.

Specification Number: 42 Function Number: 0x0302

Parameters

bioPeripheral	The handle to the event source associated with the target Biometric Peripheral.
level	The assurance level at which to capture the biometric data.
tag	A pointer to the tag used to identify this operation, in events or in the TEE_BioStop function. See section 4.6.2.1.

Return Value

TEE_SUCCESS	In case of success.
TEE_ERROR_NO_DATA	If there are no enrolled templates.
TEE_ERROR_OUT_OF_MEMORY	If the system ran out of resources.
TEE_ERROR_BUSY	If the TUI resources are currently in use.

Panic Reasons

- If bioPeripheral does not refer to a Biometric Peripheral.
- If tag pointer is NULL.
- If the biometric sub-system fails to load any critical code component.
- If the Implementation detects any error associated with this function which is not explicitly associated with a defined return code for this function.

4.6.2.3 TEE_BioStartEnroll

```
TEE_Result TEE_BioStartEnroll(
[in]   TEE_EventSourceHandle   bioPeripheral,
[in]   uint32_t                displayNumber,
[out]  TEE_BioTag*             tag
);
```

Description

The `TEE_BioStartEnroll` function tells the Biometric Sub-system to start enrolling a new template using biometric input from the Biometric Peripheral referenced by `bioPeripheral`.

The function is designed to return promptly and report further progress through events. The prompt return does not indicate that the user presented a biometric feature or that the biometric data could be captured.

The operation is cancellable with the `TEE_BioStop` function. If the operation was cancelled, an event will be returned to signal cancellation and no further events will be generated by this operation.

During the enroll process, the Biometric Peripheral will open a session on the selected display and lock any required peripherals. This session may be a TUI session or an REE session; in either case, the calling TA must close any conflicting TUI sessions of its own before calling Enroll. The display is identified by the display number, and it is for the TEE or REE components of the Biometric Sub-system to make sure that the enrollment prompts appear in the correct location. The Biometric Sub-system will provide the user interface to guide the user through the enroll process. While the enrollment process is active, the Biometric Sub-system will intercept events from the peripherals needed by the Biometric Sub-system.

If a recoverable problem is encountered during the Enroll process, the Biometric Peripheral will handle this, for example by displaying instructions on display. Such errors are not reported to the TA.

While an Enroll operation is ongoing, the Biometrics API and the TUI Low-level API will act as though another TA currently has ownership of the required peripherals.

If the operation is terminated by a fatal error, an error status will be returned through the event queue. For a full list of events that may be returned, see section 4.3.5.

When the Enroll operation is complete, the Biometric Sub-system will return control of the display to the TA and issues a `TEE_EVENT_BIO_ENROLL_COMPLETED` event. The display will be left unchanged from the latest status as displayed by the Enroll operation. As soon as the TA receives a `TEE_EVENT_BIO_ENROLL_COMPLETED` event or an event indicating a fatal error, it SHOULD update the display, either by opening a new TUI session or by using the REE through its Client Application.

Expected Events

`TEE_EVENT_BIO_ENROLL_READY`	Sent when the Biometric Peripheral is ready to accept input. Do not prompt the user for input until this event has been received.
`TEE_EVENT_BIO_ENROLL_STARTED`	Sent when user first presents their biometric feature; information only.
`TEE_EVENT_BIO_ENROLL_COMPLETED`	Sent when capture is complete; contains `TEE_TemplateID` to the created template.

Failure Events

TEE_EVENT_BIO_ERROR	An error or warning has occurred.
TEE_EVENT_BIO_STOPPED	The operation was stopped using the TEE_BioStop function.

Specification Number: 42 Function Number: 0x0303

Parameters

bioPeripheral	The handle to the event source associated with the target Biometric Peripheral.
displayNumber	The display number used by [TEE TUI Low] to describe this display. The Biometric Sub-system will display instructions to the user on this display.
tag	A pointer to the tag used to identify this operation, in events or in the TEE_BioStop function. See section 4.6.2.1.

Return Value

TEE_SUCCESS	In case of success.
TEE_ERROR_NO_DATA	If there are no enrolled templates.
TEE_ERROR_OUT_OF_MEMORY	If the system ran out of resources.
TEE_ERROR_BUSY	If a TA has an open TUI session on the selected display, or if the Biometric Sub-system cannot open the required peripherals.

Panic Reasons

- If bioPeripheral does not refer to a Biometric Peripheral.
- If tag pointer is NULL.
- If the biometric sub-system fails to load any critical code component.
- If the Implementation detects any error associated with this function which is not explicitly associated with a defined return code for this function.

4.6.2.4 TEE_BioStartVerify

```
TEE_Result TEE_BioStartVerify (
[in]       TEE_EventSourceHandle bioPeripheral,
[in]       TEE_BioHandle         data,
[inbuf]    TEE_TemplateID*       templateIds, uint32_t size,
[out]      uint32_t*             tag
);
```

Description

The `TEE_BioStartVerify` function starts the process of verifying input captured with `TEE_BioStartCapture` against the previously associated templates selected through the `templateIds` input buffer using the Biometric Peripheral referenced by the `bioPeripheral`.

Information on the progress of the operation is sent via events. The TA must process these using the `TEE_Event_Wait` function.

Successful verification results in a `TEE_EVENT_BIO_VERIFY_MATCH` event. However, if the Biometric Sub-system checks all the supplied templates and does not find a match, it generates a `TEE_EVENT_BIO_VERIFY_NO_MATCH` event.

The operation is cancellable with the `TEE_BioStop` function. If the operation was cancelled, an event will be returned to signal cancellation, and no further events will be generated by this operation.

If the operation is terminated by an error, an error status will be returned through the event queue. For a full list of events that may be returned, see section 4.3.5.

The function is designed to return promptly and report further progress through events. It does not mean that the user presented their biometric feature or that the biometric data could be verified against an existing template.

While the operation is active, the Biometric Sub-system will intercept events from the peripherals needed by the Biometric Sub-system.

For details, see the relevant sequence diagram in Annex C.

Expected Events

`TEE_EVENT_BIO_VERIFY_STARTED`	Sent to confirm that the Biometric Peripheral is about to perform a Verify operation; information only.
`TEE_EVENT_BIO_VERIFY_NO_MATCH`	Sent when Verify operation does not find any match.
`TEE_EVENT_BIO_VERIFY_MATCH`	Sent when Verify operation finds a match; contains the `TEE_TemplateId`.

Failure Events

`TEE_EVENT_BIO_ERROR`	An error or warning has occurred.
`TEE_EVENT_BIO_STOPPED`	The operation was stopped using the `TEE_BioStop` function.

Specification Number: 42 Function Number: 0x0304

Parameters

`bioPeripheral`	The handle to the event source associated with the target Biometric Peripheral.
`data`	A pointer to an opaque handle to biometric data.
`templateIds, size`	An input buffer containing previously associated Template IDs to be used for the verification. The buffer can optionally be set to `NULL` with `size` set to `0`, meaning the verification should be performed with all previously associated templates.
`tag`	A pointer to the tag used to identify this operation, in events or in the `TEE_BioStop` function. See section 4.6.2.1.

Return Value

`TEE_SUCCESS`	In case of success.
`TEE_ERROR_NO_LIVE_TEMPLATE`	If the Live Template referenced by `data` is no longer available.
`TEE_ERROR_OUT_OF_MEMORY`	If the system ran out of resources.
`TEE_ERROR_BUSY`	If the TUI resources are currently in use.

Panic Reasons

- If `bioPeripheral` does not refer to an event source.
- If tag pointer is `NULL`.
- If the biometric sub-system fails to load any critical code component.
- If the Implementation detects any error associated with this function which is not explicitly associated with a defined return code for this function.

4.6.2.5 TEE_BioStop

```
TEE_Result TEE_BioStop(
[in]   TEE_EventSourceHandle      bioPeripheral,
[in]   uint32_t                   tag
);
```

Description

The `TEE_BioStop` function stops the process identified by `bioPeripheral` and `tag`.

If this is a capture process, the Biometric Peripheral will not expect further user interaction, and it can free any resources it has allocated and return control of the peripherals and event queue to the calling TA.

If the process is active, it responds by sending a `TEE_EVENT_BIO_STOPPED` event.

If the process has already completed, the function has no effect. However, it is not an error to send multiple `TEE_BioStop` functions.

Other processes running on the Biometric Peripheral are unaffected.

The Biometric Peripheral is still reserved for use by the calling TA until it is removed from the event queue.

No further events will be generated associated with this tag. However, events already issued will still need to be processed by the TA. They will not be cleared from the queue.

Because a biometric operation can complete or be cancelled asynchronously, it is not an error to call the function with a `tag` that is not valid; in this case, the function SHALL have no effect.

Specification Number: 42 Function Number: 0x0305

Parameters

`bioPeripheral`	The handle to the event source associated with the target Biometric Peripheral.
`tag`	The tag used to identify this operation. See section 4.6.2.1.

Return Value

`TEE_SUCCESS`	In case of success. Function has been processed. It is not an error to send multiple stop functions, and the TEE is not required to track the state of completed functions, thus this will be returned even if the tag is invalid.
`TEE_ERROR_OUT_OF_MEMORY`	If the system ran out of resources. Unable to process the `TEE_BioStop` function and therefore events associated with the tag may continue to be generated.

Panic Reasons

- If `bioPeripheral` does not refer to an event source.
- If the Implementation detects any error associated with this function which is not explicitly associated with a defined return code for this function.

Annex A Panicked Function Identification

If this specification is used in conjunction with the TEE TA Debug Specification ([TEE Debug]), then the specification number is 42 and the following values SHALL be associated with the function declared.

Table A-1: Function Identification Values

Category	Function	Function Number in hexadecimal	Function Number in decimal
Functions that do not require a TUI session			
	TEE_BioDissociateTemplate	0x0201	513
	TEE_BioListTemplates	0x0202	514
Functions that require a Low-level TUI session			
	TEE_BioStartAssociate	0x0301	769
	TEE_BioStartCapture	0x0302	770
	TEE_BioStartEnroll	0x0303	771
	TEE_BioStartVerify	0x0304	772
	TEE_BioStop	0x0305	773

Annex B Biometrics API Usage

The following example code is informative. No guarantee is made as to its quality or correctness.

```
#include "tee_internal_api.h"
#include "tee_tui_low_api.h"
#include "tee_tui_bio_api.h"

#define TA_GETPERIPHERALS (1)
#define TA_VERSIONFAIL    (2)
#define TA_GETSTATETABLE  (3)
#define TA_FAILBAUDRATE   (4)
#define TA_FAILOPEN       (5)
#define TA_FAILWRITE      (6)
#define TA_OPENFAIL       (7)
#define TA_NOHANDLE       (8)

struct Ehandle{
    Ehandle *next;
    uint32_t component;
    bool locked;
} ;

struct Template{
    Template *next;
    uint32_t Tid;
} ;

struct Ehandle *rootcomp;
uint32_t lastTag =0;
uint32_t numTemplates = 0;
struct Template *roottemplate;
uint32_t       i;                   // counter


TEE_Result getImage(const char* imagename, TEE_TUIImage* foundimg) {
    // helper function to find the ObjectID for an image from its name.

  if (sizeof(imagename) == 0){
    foundimg = 0;
    return TEE_ERROR_ITEM_NOT_FOUND;
  }
  else{
    // code to find image and set foundimg would go here
    return TEE_SUCCESS;
  }
}
```

```
TEE_Result tuiLowLevel() {
/* This is an example of how a low level API may make use of biometrics
 * A call to this function will open a standard low level interface with
 * the addition of giving the user the option to provide a biometric
 * If a biometric is provided then the function looks for a match and
 * performs an action if it finds one
 */

  TEE_EventQueueHandle      *myQueue;             // points to event queue
  TEE_PeripheralId           myFingerId;          // Id for chosen fingerprint scanner
  TEE_PeripheralId           myTouchId;           // Id for Touch event source for display
  TEE_PeripheralId           myTeeId;             // Id for Event Source for TEE events
  TEE_PeripheralDescriptor  *myFingerPD;          // Peripheral Descriptor
  TEE_PeripheralDescriptor  *myTouchPD;           // Peripheral Descriptor
  TEE_PeripheralDescriptor  *myTeePD;             // Peripheral Descriptor
  TEE_PeripheralDescriptor   myPDArray[3];        // Array of Peripheral Decriptors
  TEE_EventSourceHandle      myFingerESH;         // Handle for fingerprint scanner
  TEE_EventSourceHandle      myTouchESH;          // Handle for Touch event source
                                                      //  associated with display
  TEE_EventSourceHandle      myTeeESH;            // Handle for Event Source for events
  TEE_EventSourceHandle      myDeviceArray[2];    // array of event sources
  TEE_TUIImage              *myPng;               // pointer to image to show
  uint32_t                  *myWidth;             // height of image
  uint32_t                  *myHeight;            // width of image
  uint32_t                  *myColortype;         //
  TEE_TUIDisplay            *myDisplay;           // diplay to use
  TEE_TUISurface            *myDisplaySurface;    // surface for the display
  TEE_TUIDisplayInfo        *myDisplayInfo;       // info about display
  TEE_TUISurfaceInfo         *mySurfaceInfo;      // info about surface
  int                        numSources = 2 ;     // myTeeESH myTouchESH
  uint32_t                   version  =  1;       // all structures are
                                                  // currently version 1
  bool                       match  =  false;     // indicates successful match
  bool                       complete = false;    // used in while
  uint32_t                  *numevents;           // number of events
  TEE_Event                  events[20];          // stores returned events
  TEE_Event_TUI_Touch        myTouchEvent;        // stuct for touch events
  TEE_Event_Bio              myBioEvent;          // strut for bio events
  uint32_t                  *dropped;             // count of dropped events.
  uint32_t                   myTemplate[0];       // list of acceptable templates
  TEE_BioTag                *myTag;               // tag to identify commands
  TEE_TemplateID             matched;             // specific template matched
  TEE_Peripheral             myPeripherals[1];    // array of peripherals
  TEE_Result                 myResult;
  size_t                     size;
  uint32_t                   max;
  bool                       supports_exclusive;
  bool                       supports_baudrate_change;
  uint8_t                    buf[256];
  size_t                     bytes_in_Peripheral_array = 0;
  size_t                     bytes_in_state_table = 0;
  TEE_Peripheral             peripherals[1];
  TEE_PeripheralState        myPeripheralstate[1];

     // Trivial error handling
  #define ta_assert(cond, val); if (!(cond)) TEE_Panic(val);
```

```
   // find out how much space is needed for information on the available peripherals
  TEE_Peripheral_GetPeripherals(&version, NULL, &bytes_in_Peripheral_array) !=
TEE_ERROR_SHORT_BUFFER

  // defend against stange result
  ta_assert((myResult == TEE_ERROR_SHORT_BUFFER) && (bytes_in_Peripheral_array != 0),
            TA_GETPERIPHERALS);

  peripherals = TEE_Malloc( bytes_in_Peripheral_array, TEE_MALLOC_FILL_ZERO);

   // fetch the information on the peripherals

  myResult = TEE_Peripheral_GetPeripherals(&version, peripherals, &bytes_in_Peripheral_array);

   // check result is sensible
  ta_assert((myResult == TEE_SUCCESS) && (&bytes_in_Peripheral_array != 0),
            TA_GETPERIPHERALS);

   // assume we only have one display
  // if the device has more displays, the TA must determine which to use.
  // find touch source for display 1

  if (TEE_TUI_GetDisplayInformation(1,
                                    &version,
                                    myDisplayInfo,
                                    myDisplay) != TEE_SUCCESS) {

      /* error obtaining Display Info buffer */
  }

  //*****************************************************************
  // Find Peripheral ID for OS pseudo-peripheral (there is only one) and for
  // the proprietary UART (there is also only one, for simplicity)
  //*****************************************************************

  max = size / sizeof(TEE_Peripheral);
  for (uint32_t i = 0; i < max; i++) {
    ta_assert(peripherals[i].version == 1, TA_VERSIONFAIL);
    if (peripherals[i].u.v1.periphType == TEE_PERIPHERAL_OS) {
      myTeePD->id = peripherals[i].u.v1.id;
    }

    // find fingerprint source

    if (myPeripherals[i].u.v1.periphType == TEE_PERIPHERAL_BIO) {
         size = sizeof(myPeripheralstate);
         myResult = TEE_Peripheral_GetStateTable(peripherals[i].u.v1.id,
                    myPeripheralstate, &size);
         ta_assert((myResult == TEE_SUCCESS) && (size <= sizeof(myPeripheralstate)),
                   TA_GETSTATETABLE);
         // Get a value from TEE pseudo-peripheral
         max = size / sizeof(TEE_PeripheralState);
         for (uint32_t j = 0; j < max; j++) {
              if ((myPeripheralstate[j].u.uint32val == TEE_BIO_FINGERPRINT) &&
             (myPeripheralstate[j].id == TEE_BIO_TYPE)) {
                       myFingerPD->id = peripherals[i].u.v1.id;
              };
         };
    };
```

```
  // find touch source

  if (myPeripherals[i].u.v1.periphType == TEE_PERIPHERAL_TOUCH) {
  for (uint32_t j = 0; j < myDisplayInfo->numPeripherals; j++){
       if (myPeripherals[i].u.v1.periphType == myDisplayInfo->associatedPeripherals[j]){
            myTouchPD->id = peripherals[i].u.v1.id;
                 }
       }
  }
  }

  // open the display and peripherals.
  myPDArray[0] = *myTeePD;
  myPDArray[1] = *myFingerPD;
  myPDArray[2] = *myTouchPD;
  myResult = TEE_TUI_InitSessionLow(1,1000,
            TEE_TUI_INIT_SESSION_LOW_FLAGS_REQUIREINDICATOR,
            myDisplay, 3, myPDArray);
  switch (myResult){
       case TEE_SUCCESS:
            myTeeESH =    myPDArray[0].eHandle;
            myFingerESH = myPDArray[1].eHandle;
            myTouchESH =  myPDArray[2].eHandle;
            break;

       case TEE_ERROR_BUSY:
            // Touch event source is in use return an error
            return TEE_ERROR_BUSY;

       default:
            // some other error so assert
            TEE_Panic(myResult);
  }

  if (TEE_TUI_GetDisplaySurface(myDisplay,
                            myDisplaySurface) != TEE_SUCCESS) {

     /* Error opening Display Surface */
  }

  /* Get a new surface to use to manipulate the image */

if (TEE_TUI_GetSurfaceInformation(&version,
                                  myDisplaySurface,
                                  mySurfaceInfo) != TEE_SUCCESS) {

     /* error obtaining Surface Info*/
}

if (TEE_TUI_GetSurface(myDisplayInfo->pixelWidth,
                       myDisplayInfo->pixelHeight,
                       mySurfaceInfo->u.v1.stride,
                       myDisplaySurface) != TEE_SUCCESS) {
     /* error obtaining working buffer */
}

 myDeviceArray[0] = myTeeESH;
 myDeviceArray[1] = myTouchESH;
```

```
if (TEE_Event_OpenQueue(&version,
                        numSources,
                        1000,
                        myDeviceArray,
                        myQueue
                        ) != TEE_SUCCESS) {
 /* Cannot open Event queue */
}

if (TEE_BioStartCapture(myFingerESH, TEE_BIO_ASSURANCE_HIGH, myTag) != TEE_SUCCESS) {
      // Cannot  open BioDevice
}

/* Wait for an event */

while (!complete) {
    if (TEE_Event_Wait(myQueue,
                       100,
                       events,
                       numevents,
                       dropped) != TEE_SUCCESS) {
          /* deal with error */
    } else {

        /* deal with input */
        for(int idx=0;idx < *numevents; idx++)
        {
            switch (events[idx].u.v1.eventType) {
          case TEE_EVENT_TYPE_TUI_TOUCH:
            /* User touched display work out which button was pressed */

            memcpy(&myTouchEvent,
                   events[idx].u.v1.payload,
                   sizeof(myTouchEvent));

            /* MyTouchEvent contains a touch */
            /* Code to deal with touch events goes here */
            break;

          case TEE_EVENT_TYPE_BIO:
               /* we have a bio event */
            memcpy(&myBioEvent,
                   events[idx].u.v1.payload,
                   sizeof(myBioEvent));

            // if we had multiple open biometric components we would need to check
            // events[idx].handle
            // if we had multiple commands running we would need to check myBioEvent.tag
            // we could assert myBioEvent.tag == myTag

            switch (myBioEvent.u.v1.bioType) {
              /* based on the event type we select the image to display
                 we then update the display */

              case TEE_EVENT_BIO_CAPTURE_READY:
                 /* Ready for input - tell user to swipe finger */
                if (getImage("ready.png", myPng) != TEE_SUCCESS) {
                  // error finding image
                  }
                break;
```

```
                case TEE_EVENT_BIO_CAPTURE_STARTED:
                  /* User has presented their finger */
                  if (getImage("swipe.png",  myPng) != TEE_SUCCESS) {
                    // error finding image
                    }
                  break;
                case TEE_EVENT_BIO_ERROR:
                case TEE_EVENT_BIO_CAPTURE_INSUFFICIENT_DATA:

                  /* Cannot complete capture   */
                  /* myBioEvent.data.error explains why */
                  if (getImage("sorry.png",  myPng) != TEE_SUCCESS) {
                    // error finding image
                    }
                  break;

                case TEE_EVENT_BIO_CAPTURE_COMPLETED:
                  /* Capture completed successfully
                  /* Start verification */

                  if (TEE_BioStartVerify(myFingerESH, myBioEvent.u.v1.data.handle,
                        myTemplate, sizeof(myTemplate), myTag) != TEE_SUCCESS) {
                     // Cannot start verification process
                     if (getImage("sorry.png", myPng) != TEE_SUCCESS) {
                        // error finding image
                        }
                     }
                  else {
                     /*  Starting to verify - tell user */
                     if (getImage("starting_to_verify.png", myPng) != TEE_SUCCESS) {
                        // error finding image
                     }
                  }
                    break;

                case TEE_EVENT_BIO_VERIFY_MATCH:
                  /* matched template */
                  if (getImage("matched.png",  myPng) != TEE_SUCCESS) {
                    // error finding image
                    }
                  complete = true;
                  match  =  true;
                  matched = myBioEvent.u.v1.data.templateId;
                  break;

                case TEE_EVENT_BIO_VERIFY_NO_MATCH:
                  /* Finger does not match MyBioEvent.template */
                  if (getImage("failed.png", myPng) != TEE_SUCCESS) {
                  // error finding image
                    }
                  complete = true;
                  match = false;
                  break;
                 } // end of switch on bio event type
              // update display with new image
              /* Obtain the size of the image */
              if (TEE_TUI_GetPNGInformation(myPng,
                                            myWidth,
                                            myHeight,
                                            myColortype) != TEE_SUCCESS) {
                /* error obtaining information from image */
                }
              // write it  to buffer
```

```
            if (TEE_TUI_SetPNG(myDisplaySurface,
                        myPng,
                        0,
                        0,
                        myWidth,
                        myHeight,
                        TEE_TUI_SURFACE_OPACITY_OPAQUE) != TEE_SUCCESS) {
              /* error writing PNG to surface */
              }
            /* Call BlitDisplaysurface to update display */
            if (TEE_TUI_BlitDisplaySurface(myDisplay, myDisplaySurface) != TEE_SUCCESS){
              /* error updating display */
              }
            // get new surface
            if (TEE_TUI_GetDisplaySurface(myDisplay,
                                          myDisplaySurface) != TEE_SUCCESS) {
              /* error obtaining surface for display */
              }
            break;

            } // end of switch
          } // end of for
      } // end of else
  }  // end of while
  return TEE_SUCCESS;
}

int main(){
    TEE_Result r;
    rootcomp->next = 0;
    r = tuiLowLevel();
    }
```

Annex C Sequence Diagrams

Figure C-1: Biometric Enroll Using TUI Low-level API

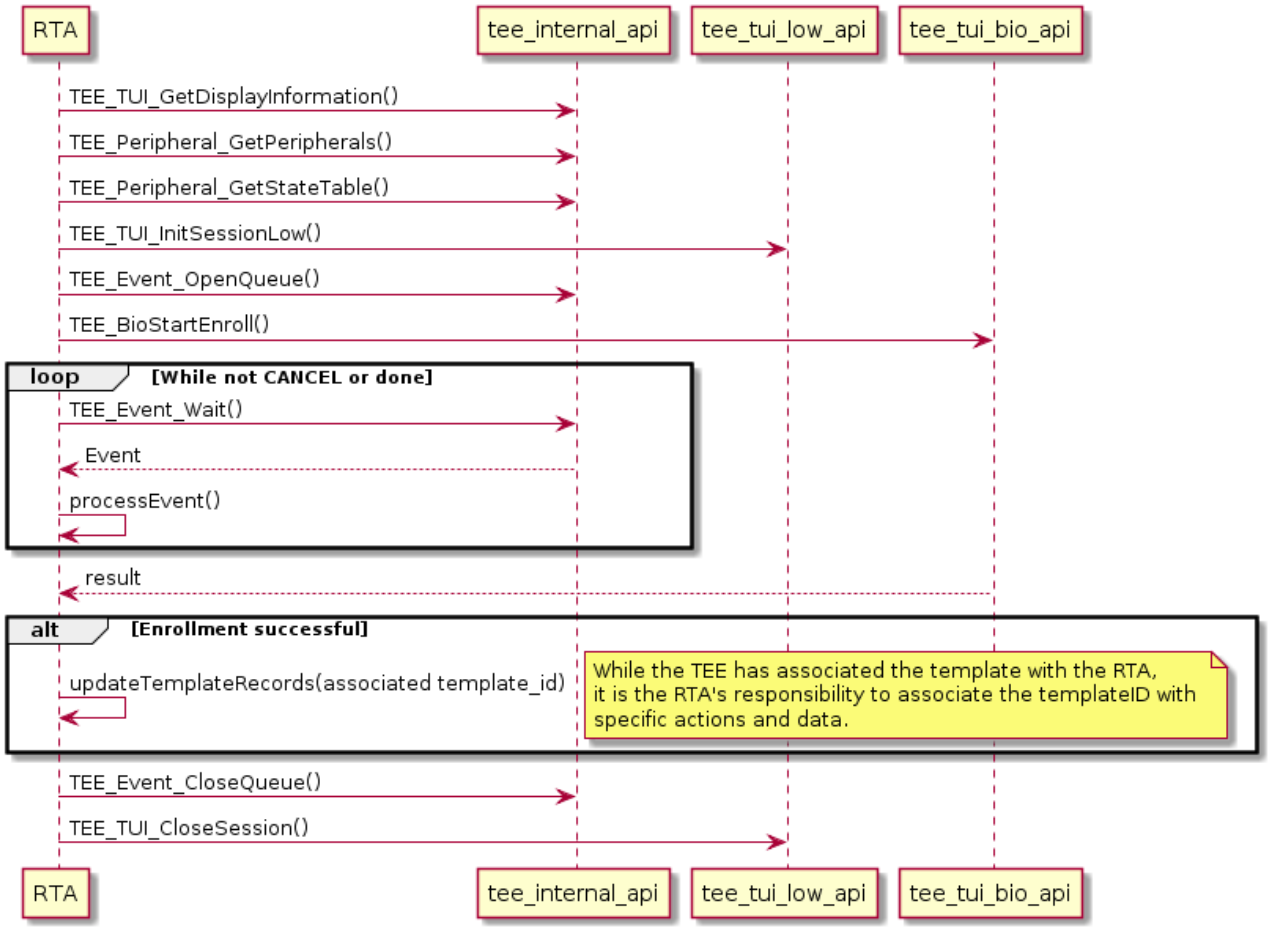


Figure C-2: Biometric Associate Using TUI Low-level API

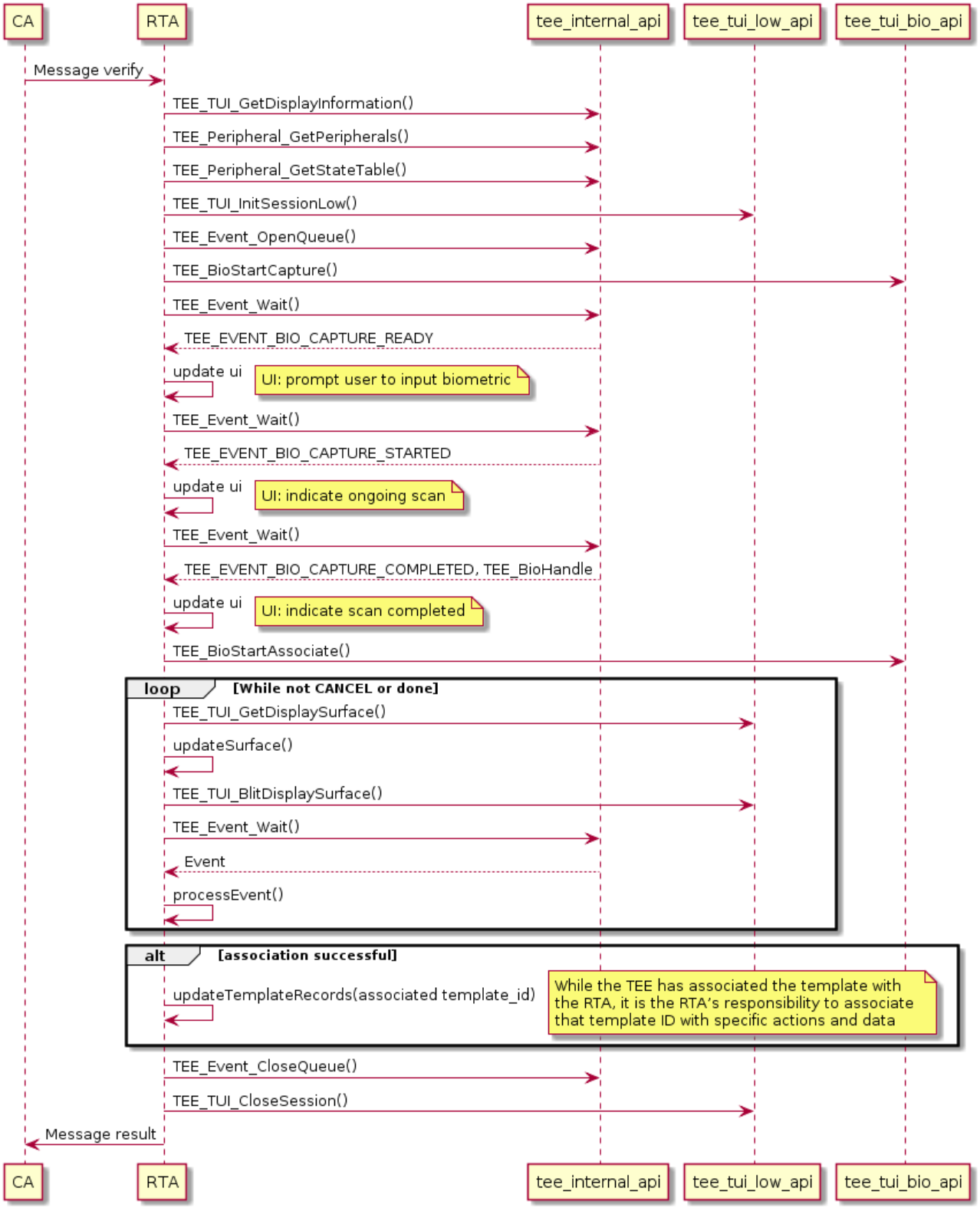


Figure C-3: Biometric Verify Using TUI Low-level API