GlobalPlatform Device Technology

Trusted User Interface API

Version 1.0

Public Release

June 2013

Document Reference: GPD_SPE_020

This page intentionally left blank.

Contents

1 Introduction 7
1.1 Audience 7
1.2 IPR Disclaimer 7
1.3 References 7
1.4 Terminology and Definitions 8
1.5 Abbreviations and Notations 9
1.6 Conventions 10
1.7 Revision History 10

2 Trusted User Interface Objectives 11
2.1 Target 11
2.2 Purpose 11
2.3 Functionalities and Use Cases 12
2.3.1 PIN Entry Functionality 12
2.3.2 Login / Password Entry Functionality 12
2.3.3 Message Functionality 12
2.4 External Source for Input of Displayed Message 13
2.4.1 Remote Servers 13
2.4.2 Simple Remote Terminal Devices 13
2.4.3 Device Local Messages 13

3 Trusted User Interface Principles 14
3.1 Overall Architecture 14
3.2 Trusted User Interface Screens 15
3.2.1 PIN Entry Screen 16
3.2.2 Login and Password Entry Screen 17
3.2.3 Message Screen 18
3.3 Authorized Button Combinations 20
3.4 Label Structure 21
3.5 Security Indicator 22
3.6 TUI Session 22
3.7 Images Format 23
3.8 Minimum Number of Input Fields 23
3.9 Input Text 24
3.9.1 Input Field Alphabet 24
3.10 Output Text 24
3.10.1 Default Alphabet 24
3.10.2 Supported Languages 24
3.10.3 Format 25
3.10.4 Minimum Text Area for Screen Labels 25
3.10.5 Markup Capabilities and Text Adjustment 25
3.11 Power and OS Events Management 26
3.12 Screen Orientation 26
3.13 Accessibility 26

4 Trusted User Interface API 27
4.1 Implementation Properties 27
4.2 Header File 27
4.3 Data Constants 28
4.4 Data Types 29
4.4.1 TEE_TUIEntryFieldMode 29

4.4.2 TEE_TUIEntryFieldType 29
4.4.3 TEE_TUIScreenOrientation 30
4.4.4 TEE_TUIButtonType 30
4.4.5 TEE_TUIImageSource 31
4.4.6 TEE_TUIImage 31
4.4.7 TEE_TUIScreenLabel 32
4.4.8 TEE_TUIButton 33
4.4.9 TEE_TUIScreenConfiguration 34
4.4.10 TEE_TUIScreenButtonInfo 35
4.4.11 TEE_TUIScreenInfo 36
4.4.12 TEE_TUIEntryField 37
4.5 Functions 38
4.5.1 TEE_TUICheckTextFormat 38
4.5.2 TEE_TUIGetScreenInfo 39
4.5.3 TEE_TUIInitSession 39
4.5.4 TEE_TUICloseSession 40
4.5.5 TEE_TUIDisplayScreen 41
Annex A TUI API Usage 43
Annex B Panicked Function Identification 48

Figures

Figure 3-1: TEE with TUI Architecture 14
Figure 3-2: Example of a Typical PIN Entry Screen 16
Figure 3-3: Example of a Typical Login/Password Entry Screen 17
Figure 3-4: Example of a Typical Message Screen for Information 18
Figure 3-5: Example of a Typical Message Screen for Validation 19
Figure 3-6: Example of Typical Message Screens for Validation with Previous and Next Buttons 19
Figure 3-7: Label Structure 21
Figure 3-8: Label Composition Example 21

Tables

Table 1-1: Normative References 7
Table 1-2: Informative References 8
Table 1-3: Terminology and Definitions 8
Table 1-4: Abbreviations and Notations 9
Table 1-5: Revision History 10
Table 3-1: Authorized Button Combinations (Excluding CORRECTION) 20
Table 4-1: Implementation Properties 27
Table 4-2: Error Code 28
Table 4-3: Data Constants 28

1 Introduction

Many sensitive use cases lead to an interaction with the user. They are mainly, but not exclusively, related to financial services and corporate usages: bill payment, money transfer, document signature validation, privacy, etc.

While the TEE Internal Core API [TEE Internal API] offers the possibility to execute all sensitive operations within a Trusted Application (TA) running in the Trusted Execution Environment (TEE), certain applications need to expose sensitive information to the user for validation or to get sensitive information from the user. Entering a PIN and signing a document are examples of operations that need to be handled inside the TEE for the Trusted Application and not to rely on facilities in the Rich Execution Environment (REE).

This document defines and specifies a Trusted User Interface (TUI) API for Trusted Application developers.

It is not the role of this interface to provide security for streaming media display or to secure other device interfaces such as audio or camera components.

This document is an addendum to [TEE Internal API].

1.1 Audience

This document is intended to support software developers implementing Trusted Applications running inside the TEE which need to display sensitive information to the user or retrieve sensitive data from the user.

This document is also intended for implementers of the Trusted User Interface in the TEE itself.

1.2 IPR Disclaimer

GlobalPlatform draws attention to the fact that claims that compliance with this specification may involve the use of a patent or other intellectual property right (collectively, "IPR") concerning this specification may be published at https://www.globalplatform.org/specificationsipdisclaimers.asp. GlobalPlatform takes no position concerning the evidence, validity, and scope of these IPR claims.

1.3 References

Table 1-1: Normative References

Standard / Specification	Description	Ref
GPD_SPE_010	GlobalPlatform Device Technology TEE Internal API Specification	[TEE Internal API]
ISO 639-1	Codes for the representation of names of languages – Part 1: Alpha-2 code	[ISO 639-1]
PNG	ISO/IEC 15948:2004 - Information technology -- Computer graphics and image processing -- Portable Network Graphics (PNG): Functional specification	[ISO 15948]
RFC 2119	Key words for use in RFCs to Indicate Requirement Levels	[RFC 2119]
Unicode	The Unicode Standard; available at: http://www.unicode.org/versions/Unicode6.2.0/	[Unicode]

Table 1-2: Informative References

Standard / Specification	Description	Ref
GPD_SPE_007	GlobalPlatform Device Technology TEE Client API Specification	[TEE Client API]
GPD_SPE_009	GlobalPlatform Device Technology TEE System Architecture	[TEE Sys Arch]
GPD_SPE_025	GlobalPlatform Device Technology TEE TA Debug Specification	[TEE Debug]

1.4 Terminology and Definitions

Table 1-3: Terminology and Definitions

Term	Definition
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)*.
Data Object	An object containing a data stream but no key material.
Object Identifier	A variable-length binary buffer identifying a persistent object.
Panic	An exception that kills a whole TA instance as a result of calling one of the API functions.
Property	An immutable value identified by a name.
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 un-trusted. Contrast *Trusted Execution Environment (TEE)*.
Rich OS	Typically an OS providing a much wider variety of features than that of 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 the size and needs of the Rich OS it 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 has to be considered un-trusted, though from the Rich OS point of view there may be internal trust structures. Contrast *Trusted OS*.
Secure Element (SE)	A tamper resistant 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 SE, SIM, UICC, smartSD, smart microSD, etc.

Term	Definition
Session	Logically connects multiple commands invoked on a Trusted Application.
Task	The entity that executes any code executed in a Trusted Application.
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 [TEE Internal API] to Trusted Applications.
Trusted Storage	Storage accessible only to Trusted Applications.
Trusted User Interface (Trusted UI or TUI)	A user interface that ensures that the screen is controlled by the TEE and isolated from the REE and even the TAs.

1.5 Abbreviations and Notations

Table 1-4: Abbreviations and Notations

Abbreviation / Notation	Meaning
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ID	IDentifier
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
LED	Light Emitting Diode
NFC	Near Field Communication
OS	Operating System
PIN	Personal Identification Number
PNG	Portable Network Graphics
REE	Rich Execution Environment
RFC	Request For Comments; may denote a memorandum published by the IETF

Abbreviation / Notation	Meaning
RGB	Red Green Blue; an additive color model
SD	Secure Digital
SE	Secure Element
SIM	Subscriber Identity Module
TA	Trusted Application
TEE	Trusted Execution Environment
TUI	Trusted User Interface
UCS	Universal Character Set
UI	User Interface
UICC	Universal Integrated Circuit Card
UTF-8	UCS Transformation Format – 8-bit

1.6 Conventions

Throughout this document, normative requirements are highlighted by use of capitalized key words as described below.

The key words “MUST”, “MUST NOT”, “SHOULD”, “SHOULD NOT”, and “MAY” in this document are to be interpreted as described in [RFC 2119]:

MUST – This word means that the definition is an absolute requirement of the specification.

MUST NOT – This phrase means that the definition is an absolute prohibition of the specification.

SHOULD – This word means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

SHOULD NOT – This phrase means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

MAY – This word means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

1.7 Revision History

Table 1-5: Revision History

Date	Version	Description
June 2013	1.0	Initial release

2 Trusted User Interface Objectives

2.1 Target

This specification is targeted at a TEE running within a smartphone or a tablet. A Trusted User Interface 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 one touchscreen or one screen and one keyboard which MUST be wired and integral to the device. Remote peripherals are not considered in this specification.

2.2 Purpose

The Trusted User Interface API permits the display of screens to the user and achieves three objectives:

- Secure Display – Information displayed to the user cannot be accessed, modified, or obscured by any software within the REE or by an unauthorized application in the TEE.
- Secure Input – Information entered by the user cannot be derived or modified by any software within the REE or by an unauthorized application in the TEE.
- Secure Indicator – The user can be confident that the screen displayed is actually a screen displayed by a TA.

2.3 Functionalities and Use Cases

This section describes the objectives of the Trusted User Interface API in terms of functionalities. It also gives examples of the use cases foreseen to be coupled with those functionalities.

While additional functionality can be required to cover a larger set of use cases, such functions have been removed in this version of the specification. This decision was made for three reasons:

- They are not realistic technically – e.g. management of peripherals such as the voice or the camera as inputs.
- They cannot yet be easily standardized due to fragmentation – e.g. video rendering.
- The use cases they cover were not considered of sufficient importance to be included in this first version of the specifications – e.g. display of complex formats.

2.3.1 PIN Entry Functionality

The Trusted User Interface API permits the user to validate an operation by entering a numeric identifier such as a PIN. PIN security is very common and is used to authorize operations all over the world.

Many use cases are in the scope of this functionality, in particular those requiring user authentication to validate or authorize a sensitive operation.

PIN entry screens can be triggered by different use cases:

- Mobile financial services such as consulting a bank account or paying a bill from an application require correct PIN entry to verify that a user is requesting an operation.
- Payment: TUI PIN entry can be used in conjunction with Secure Element (SE) applications to reduce the risk of payments by providing a level of indication of user presence.
- Authorization to view sensitive data
- Authorization to change device state and other capabilities

2.3.2 Login / Password Entry Functionality

Entering a login and the corresponding password in a protected manner is covered by the Trusted User Interface API. While several sensitive operations can require authentication using this functionality, the definitive use case is the need to authorize access to personal account data in particular through the Internet.

2.3.3 Message Functionality

The Trusted User Interface API permits the display of sensitive information to the user and optionally enables the user to indicate acceptance or rejection of that displayed data. Such screens can be triggered by different use cases, such as:

- Room number access code
- Contract validation
- One time password token
- Medical information

2.4 External Source for Input of Displayed Message

TUI screens are displayed by the TEE on behalf of TAs. While often the backgrounds and text structuring of the TUI screens will likely be already contained by the TA itself, it is perceived that three sources can directly or indirectly customize the message to display.

2.4.1 Remote Servers

Such servers are presumed for the purpose of this API to be able to compose complicated graphic or text based messages. In this case functionality such as internationalization of the message can be performed by the remote server before download to the TA and finally the TUI.

2.4.2 Simple Remote Terminal Devices

This case covers NFC-like terminals, which will typically deliver very simple concepts (number of items, name of items, cost of items) and expect the supporting device to structure those concepts into a more user friendly interaction. In this case the TA can be expected to perform functionality such as internationalization.

2.4.3 Device Local Messages

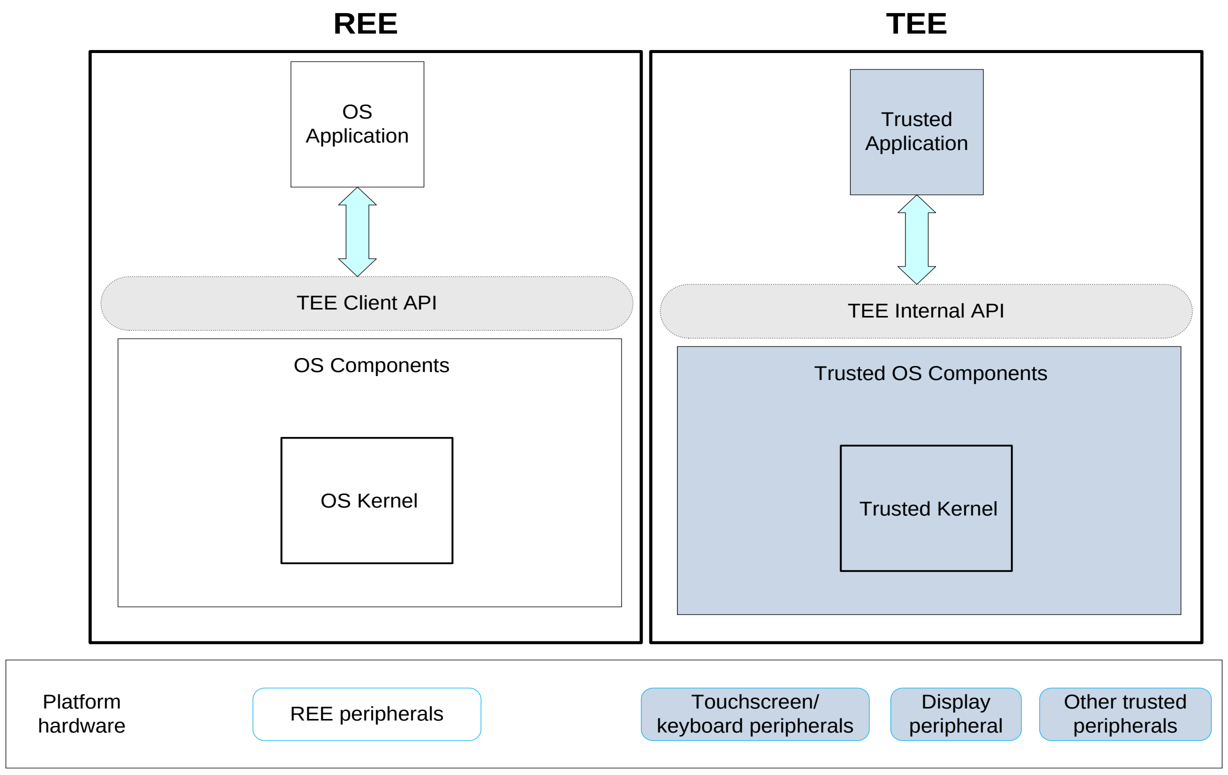
This case covers the need for services installed as basic functionality of the device to provide trusted interface capability.

This case can fall into either of the above two cases as far as the contents delivered. In some instances it can make use of complex but predefined messages that are installed during manufacturing. In other cases it can want to compose messages from simple concepts.

3 Trusted User Interface Principles

3.1 Overall Architecture

Figure 3-1: TEE with TUI Architecture



In this specification, peripherals related to the user interface MUST be wired to the device. Remote peripherals are not considered in this specification.

A typical architecture implementing the Trusted User Interface feature consists of either a touchscreen or keyboard peripheral and a display controller peripheral. When a Trusted User Interface screen is displayed, those peripherals MUST NOT be accessible for reading or writing by the REE and indication of associated events MUST NOT be received by the REE. At other times, it is up to a specific platform or a specific TEE implementation as to whether to give back the control of those peripherals to the REE or to provide some other method to allow the REE access to those peripherals.

3.2 Trusted User Interface Screens

TUI screens rely on exclusive access to UI resources from the TEE, they MUST always be displayed in foreground, and they MUST always have the focus. It is highly recommended to have the trusted part of the screen close to the full screen size and not to deal with overlays in order to avoid confusion and a bad user experience. TUI screens imply critical and sensitive data and operations that need to be immediate and exclusive.

Hereafter, typical screens managed by the Trusted User Interface API are described.

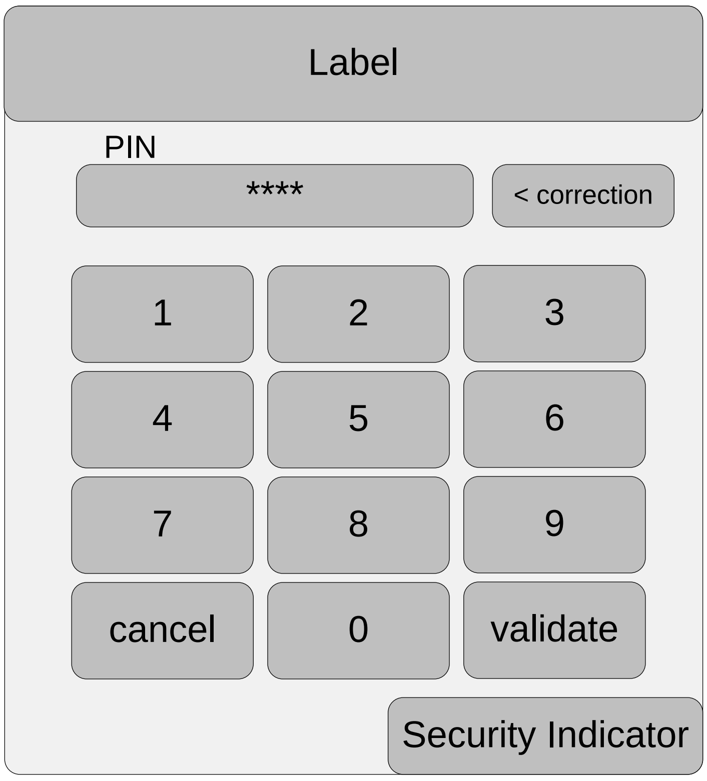
Important Notice: The following sections describe the different functionalities that MUST be supported by the TUI screens; nevertheless, the proposed figures are for reference only. For instance, if an implementation prefers to get user inputs using a physical keyboard rather than a virtual keyboard and virtual buttons, screen figures with a virtual keyboard and buttons are not accurate in that case. Likewise, labels of entry fields and texts of buttons are examples and MAY be customized by TAs.

3.2.1 PIN Entry Screen

Typically, a PIN entry screen is composed of:

- A label, holding branding information and usually detailing the operation to be validated by a PIN entry
- An entry field to indicate entered digits
- A keyboard that permits entry of digits
- Buttons:
  - o CORRECTION which allows the digit previously entered to be corrected
  - o CANCEL which allows the operation to be canceled and exits the screen
  - o VALIDATE which triggers validation of the PIN entry and exits the screen
- A security indicator (text, image, LED, …)

Figure 3-2: Example of a Typical PIN Entry Screen

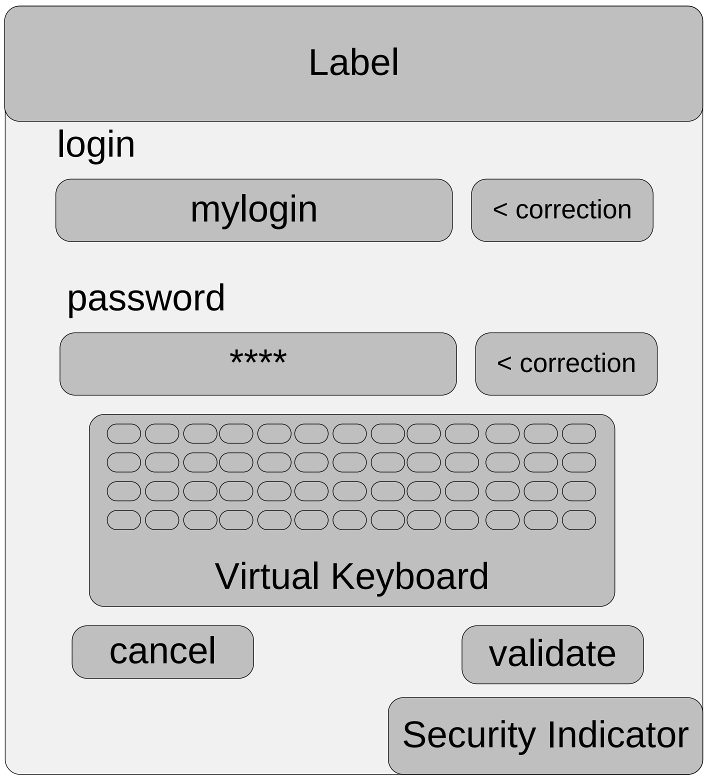


3.2.2 Login and Password Entry Screen

Typically, a login/password entry screen is composed of:

- A label, holding branding information and usually detailing the operation requiring a password
- An entry field to display the entered alphanumeric characters for the login
- An entry field to display the entered alphanumeric characters for the password
- A keyboard that permits entry of alphanumeric characters
- Buttons:
  - o CORRECTION which allows the digit previously entered to be corrected
  - o CANCEL which allows the operation to be canceled and exits the screen
  - o VALIDATE which triggers validation of the field entries and exits the screen
- A security indicator (text, image, LED, …)

Figure 3-3: Example of a Typical Login/Password Entry Screen



3.2.3 Message Screen

Typically, a message screen is composed of:

- A label area containing the message to be displayed to the user and any associated branding
- Depending on the usage, up to five buttons:
  - OK which exits the screen and informs the TA of the button ID
  - CANCEL which exits the screen and informs the TA of the button ID
  - VALIDATE which exits the screen and informs the TA of the button ID
  - NEXT which exits the screen and informs the TA of the button ID. It requires the TA to display another screen which is the continuation of the current displayed screen.
  - PREVIOUS which exits the screen and informs the TA of the button ID. It requires the TA to display the screen previously seen.
- A security indicator (text, image, LED, …)

Figure 3-4: Example of a Typical Message Screen for Information

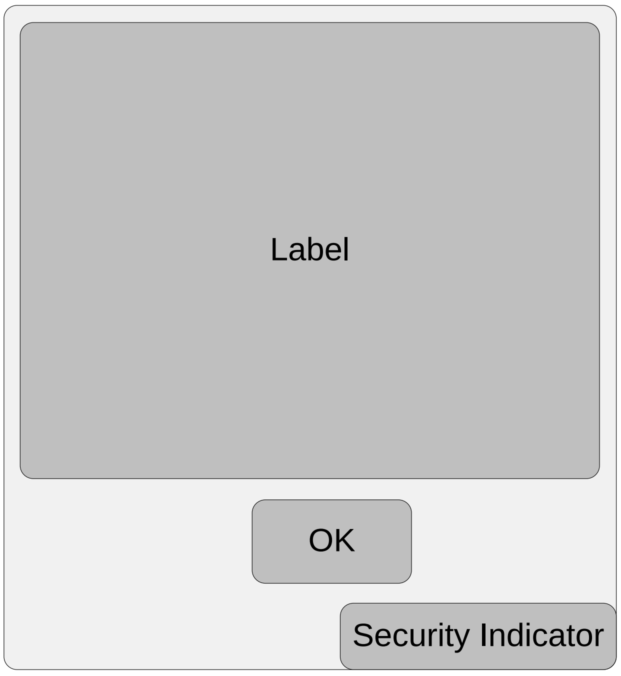


Figure 3-5: Example of a Typical Message Screen for Validation

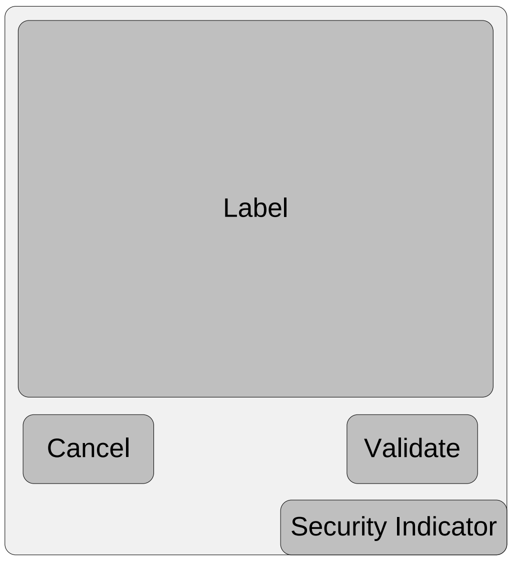
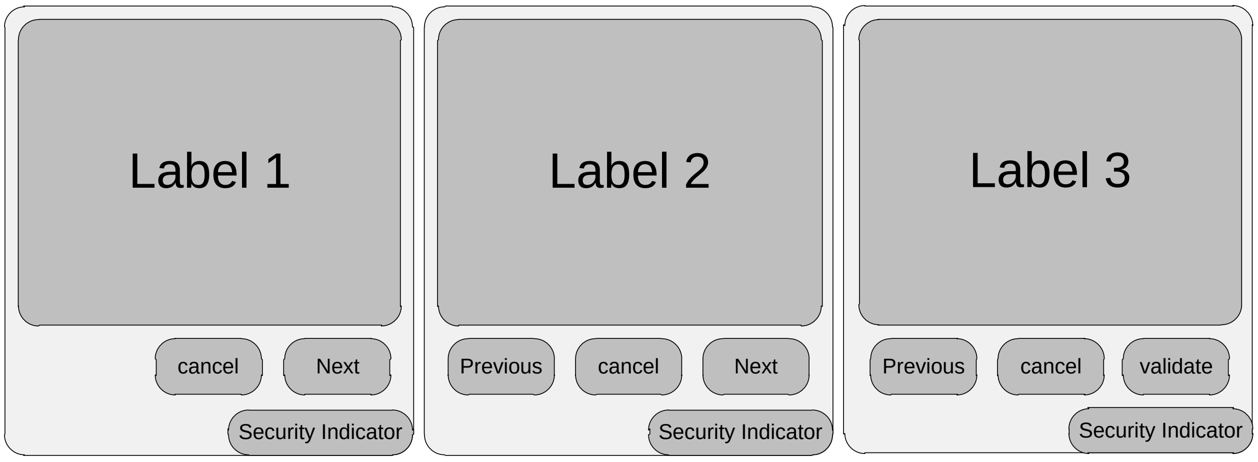


Figure 3-6: Example of Typical Message Screens for Validation with Previous and Next Buttons



3.3 Authorized Button Combinations

As described in the previous sections, six buttons are managed by the TUI API: CORRECTION, OK, CANCEL, VALIDATE, PREVIOUS, and NEXT.

The CORRECTION button is mandatory and MUST be available when at least one entry field is present. For the other cases, Table 3-1 described the authorized combinations:

Table 3-1: Authorized Button Combinations (Excluding CORRECTION)

OK	CANCEL	VALIDATE	PREVIOUS	NEXT
X				
X			X	
	X	X		
	X			X
	X		X	X
	X	X	X	

3.4 Label Structure

The label is structured from a background canvas, overlaid with two additional displays:

- A settable and discoverable overall label area canvas color
- An optional image, equal to or smaller than the canvas, can be positioned anywhere inside the Label area. Typically, it will contain the logo of a service provider.
- On top of that image can be placed optional text which is also positioned anywhere inside the Label area.

The origin point for the screen co-ordinates of image and text areas inside the label field is the top left corner of the label, with the offset value increasing from 0,0 towards the bottom right corner of the label area. These offset value indicate the top left corner of the relevant image or text area.

The combination of these three areas gives a simple but flexible display area.

Figure 3-7: Label Structure

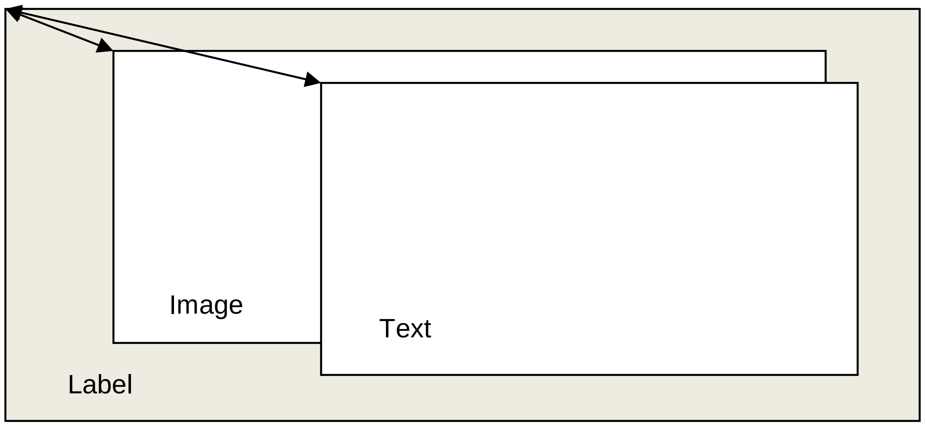
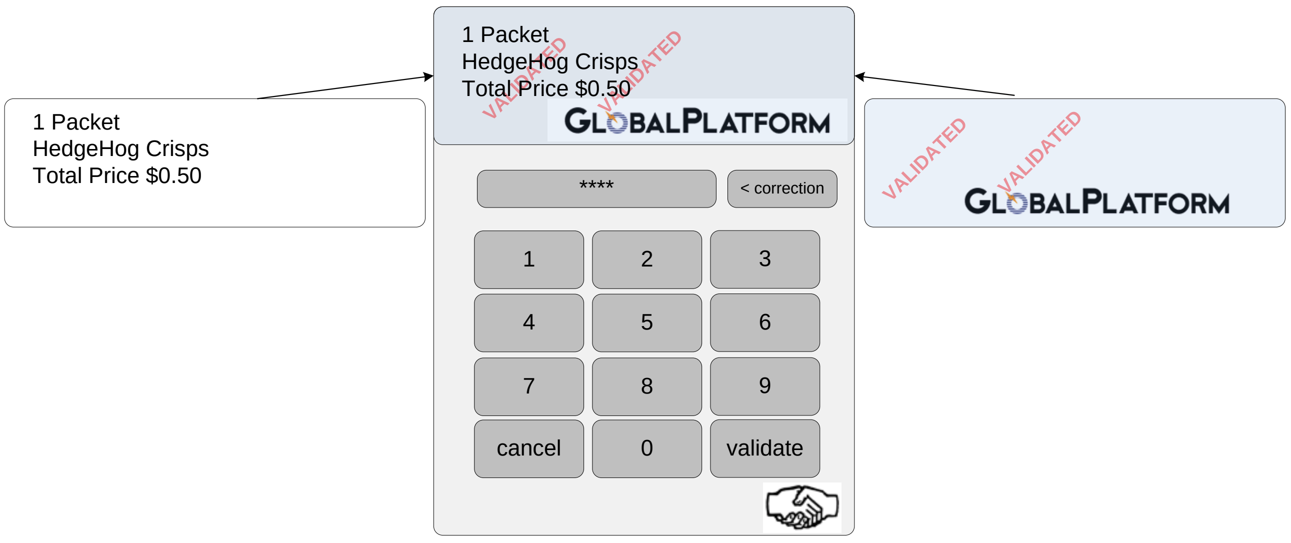


Figure 3-8: Label Composition Example



3.5 Security Indicator

The security indicator is a specific indication that the displayed screen can be considered trusted by the user, i.e. that the screen is controlled by the TEE and isolated from the REE and even the TAs. It can be either or both of:

- A hardware controlled security indication such as an LED state or the use of an area of screen under permanent TEE control.
- A piece of personal information only known by the user such as an image or a personal question with the corresponding answer. This information MUST NOT be known or accessible by the REE. It is preferable that this security indicator is uniquely associated with a user and not with a device.

It is highly recommended that the security indicator be managed directly by the TEE itself: The TEE SHOULD provide by default a security indicator when a TUI screen is displayed. If it is not the case, then the property value `gpd.tee.tui.securityIndicator` is set to `false`, the functionality of the security indicator MUST be provided by the TA itself either through a hardware controlled peripheral if available (and if a specific TEE implementation allows that) or through the label information of the TUI screen. In that last case, the image and/or the text of the label MUST be such as could reasonably be expected to make the user confident that the displayed screen can be trusted, i.e. that it is displayed by the TEE.

3.6 TUI Session

When a TA requests a TUI screen to be displayed, its access to the UI resources MUST be exclusive and this means it is not possible to display several TUI screens simultaneously. A session mechanism permits a TA to reserve exclusive access to TUI resources and in particular to guarantee that a particular sequence of TUI screens is atomic. This screen ownership atomicity is limited by overriding events such as described in section 3.11.

When the TUI resources are reserved by a TA, that MUST NOT interfere with the REE UI behavior. Only when the effective first actual display of a TUI screen starts does the TA take the control of the input and the output of the UI.

While there is no timeout associated with a TUI screen, there is a timeout associated with a TUI session: It applies to the time spent not displaying a TUI screen within a TUI session; it is started when the session is opened and when a TUI screen is ended.[1] If this timeout is reached, the TUI session is automatically closed. The timeout value is specified by the property `gpd.tee.tui.session.timeout`.

[1] A TUI screen is ended when, for example:

- The user presses a button.
- A programmed cancel call occurs.
- An external event such as a phone call or a Backlight Off event occurs.

3.7 Images Format

The only format of images supported by the specification is the Portable Network Graphics (PNG) format [ISO 15948]. It MUST be pre-scaled to fit inside the canvas area of the label.

While an implementation might support full features of PNG standard, in this specification the mandatory support is reduced. It MUST be at least the following:

- Two color types:
  - Grayscale (0) up to 8 bits depth
  - Truecolor (2) up to 24 bits
- Interlacing method 0 (no interlacing)
- Ancillary chunks are ignored.

3.8 Minimum Number of Input Fields

The TUI API allows the TA to customize a TUI screen by selecting the number of entry fields to display. The minimal number of input fields that SHALL be supported is two as it is mandatory to support the login/password use case. However, an implementation can specify that it supports more entry fields per screen for a given orientation, and communicate this with the TA through the function `TEE_TUIGetScreenInfo`.

3.9 Input Text

3.9.1 Input Field Alphabet

The platform MUST support input characters for an input field with at least the following subset of the characters of the ASCII table:

- Characters in the interval [Unicode (U+0020) – Unicode (U+007D)].

This subset is sufficient to match most country standards for PIN entry and login password entries.

3.10 Output Text

This section concerns the text that can be written within a label.

3.10.1 Default Alphabet

By default, a subset of the characters of the Unicode table ([Unicode]) MUST be supported by an implementation:

- Character for carriage return: Unicode (U+000D)
- Characters in the interval [Unicode (U+0020) – Unicode (U+007D)]
- TUI Markup: Unicode (U+E000) – (U+E003) as specified in Section 3.10.5

3.10.2 Supported Languages

It is specific to an implementation to define which languages are supported. The strings retained by this specification are UTF-8 based.

The property `gpd.tee.tui.languages` permits a TA to indicate which languages are supported by an implementation. This is informative only as there is no direct mapping between a language and the corresponding subset of UTF-8 characters that needs to be supported. To deal with this, the function `TEE_TUICheckTextFormat` permits the TA to know precisely which UTF-8 characters are supported by an implementation.

3.10.3 Format

It is specific to an implementation to define the fonts and the size of the text displayed to the user as long as they meet a minimum set of character display capabilities as defined in Section 3.10.1 by providing those fonts and sizes.

The text displayed on the Trusted User Interface can contain sensitive information such as monetary amounts or a text to be signed. The TA calling the user interface must be sure that the graphical rendering of the screen reflects exactly the text provided to the display function. A normal display interface can affect the rendering in two ways:

- If some Unicode characters cannot be graphically displayed, a default rendering is provided.
- If a line of text is too long to be displayed on a single screen line, the text can be truncated or displayed on several lines, with or without hyphenating the words.

This behavior is not acceptable for a Trusted User Interface and the TUI API will reject any string containing characters that cannot be rendered by an implementation. To prevent the problem of text line rendering, the TUI API requires the calling TA to cut the lines to match the screen before display. If a line is too large, the display operation SHALL be rejected.

This formatting operation can be achieved by using the TUI API which offers a way to know the exact width and height that a string will occupy using the function `TEE_TUICheckTextFormat`. It is up to an implementation to have a fixed width and height for all characters or to have a fine tuned implementation that MAY adapt width and height depending on the text to render.

3.10.4 Minimum Text Area for Screen Labels

The text area has both a minimum vertical and horizontal size. This minimum is expressed in numbers of characters of the minimal ASCII font. It MUST be at least:

- 4 lines
- 25 characters per line

If a device supports more complex fonts with a higher information density per symbol, such as many Asian Unicode fonts, then it MUST support at least 4 lines of 10 characters per line, in those fonts, in readable text.

3.10.5 Markup Capabilities and Text Adjustment

It is possible to mark up text to be in bold and or to be underlined. This markup capability is based on private Unicode characters. The same value is used to mark the starting point of a special formatting section and the ending of that formatting section. It is possible to overlap them.

- Unicode (U+E000) for bold
- Unicode (U+E001) for underline

It is possible to insert multiples of one pixel wide space or one pixel height space. This permits the TA to perform its own justification on text blocks, or to adjust to text fit against a specific background image. Justification (right or left) is the same as the one followed by the language of the text.

- Unicode (U+E002) moves the current cursor right by one pixel.
- Unicode (U+E003) moves the current cursor down by one pixel.

3.11 Power and OS Events Management

TUI screens often display critical and sensitive data and operations that need to be immediate and exclusive. They must lead to an experience that makes the user confident of the data displayed and entered. On the other hand, some events that occur on the platform in the REE, such as those related to power management or incoming calls, are also critical. This section clarifies the expected behavior of the TUI session in such a situation. The overall rule is that an individual TUI screen (in a TUI session) must be considered as an atomic operation that is valid only when it has not been interrupted.

When the following power management events occur during a TUI session, the device MUST trigger the TUI session to terminate:

- Device Reset event
- Device Turn Off event
- Sleep mode Turn On event
- Backlight Off event

When an OS specific event occurs during a TUI session, the TUI session MAY terminate. Typical OS specific events are incoming calls, calendar events, email notifications, etc. The choice to terminate a TUI session on particular OS specific events is implementation specific.

When a TUI session is terminated, the TUI screen MUST disappear from the display to make sure that the user is not confused and the control of its screen area is given back to the REE. It is acceptable for the TEE TUI to display a warning that the trusted display mode is being left before handing the display control back to the REE. The TA MAY replay the interrupted TUI screen when the REE event has been resolved. Most likely, the TA and its Client Application will decide which behavior to apply in this case.

3.12 Screen Orientation

By default, the specification allows display of a screen in a fixed manner either vertically or horizontally. An implementation MUST support one of the two operations and MAY support both of them.

Knowledge of the current orientation is not critical and can be considered as informative. It is not supplied by this API and MAY be obtained by the Client Application of the TA that wishes to display a TUI screen.

3.13 Accessibility

This specification provides limited support for accessibility considerations through use of service defined images, however as providing accessibility is an end to end problem set it is believed that existing systems will cater for current needs using methods outside of this limited scope. If a particular device design requires accessibility extensions (for example audio text or large font control) then these can always be added in a proprietary manner.

4 Trusted User Interface API

4.1 Implementation Properties

The following table defines the implementation properties regarding the TUI API.

Table 4-1: Implementation Properties

Property Name	Type	Meaning
`gpd.tee.tui.securityIndicator`	`bool`	`true` if a security indicator is present on TUI screens by default. It means that a security indicator is managed by the TEE. If `false`, a security indicator MUST be managed by the TA itself.
`gpd.tee.tui.languages`	`string`	The list of supported languages separated by "`:`". The language names are based on the ISO 639-1 codes [ISO 639-1].
`gpd.tee.tui.orientation`	`integer`	`0x00000001` if it is possible to request the display of TUI screens with portrait orientation, i.e. vertical orientation only. `0x00000002` if it is possible to request the display of TUI screens with landscape orientation, i.e. horizontal orientation only. `0x00000003` if it is possible to request the display of TUI screens with portrait or landscape orientation, i.e. vertical or horizontal orientation.
`gpd.tee.tui.session.timeout`	`integer`	Duration of the timeout associated with a TUI session in milliseconds. Typical value is around 10 seconds.

4.2 Header File

The header file for the TEE Internal API must have the name "`tee_tui_api.h`".

```
#include "tee_tui_api.h"
```

4.3 Data Constants

In addition to the error codes specified in [TEE Internal API], a new one is used throughout this specification.

Table 4-2: Error Code

Constant Name	Value
`TEE_ERROR_EXTERNAL_CANCEL`	`0xFFFF0011`

A constant is used to specify the number of button types.

Table 4-3: Data Constants

Constant Name	Value
`TEE_TUI_NUMBER_BUTTON_TYPES`	`0x00000006`

4.4 Data Types

4.4.1 TEE_TUIEntryFieldMode

```
typedef enum
{
   TEE_TUI_HIDDEN_MODE=0,
   TEE_TUI_CLEAR_MODE,
   TEE_TUI_TEMPORARY_CLEAR_MODE
} TEE_TUIEntryFieldMode;
```

The `TEE_TUIEntryFieldMode` enumeration enumerates the modes supported when displaying characters within an input entry field:

- `TEE_TUI_HIDDEN_MODE` when the displayed characters are never visible in clear.
- `TEE_TUI_CLEAR_MODE` when the displayed characters are always visible in clear.
- `TEE_TUI_TEMPORARY_CLEAR_MODE` when the displayed characters are visible for a short amount of time when entered and then are hidden.

Values in the enumeration greater than or equal to `0x8000` are reserved for implementation-defined field modes.

4.4.2 TEE_TUIEntryFieldType

```
typedef enum
{
   TEE_TUI_NUMERICAL=0,
   TEE_TUI_ALPHANUMERICAL
} TEE_TUIEntryFieldType;
```

The `TEE_TUIEntryFieldType` enumeration enumerates the possible types of entry fields:

- `TEE_TUI_NUMERICAL` when the field accepts only digits as inputs.
- `TEE_TUI_ALPHANUMERICAL` when the field accepts characters and digits as inputs.

Values in the enumeration greater than or equal to `0x8000` are reserved for implementation-defined field types.

4.4.3 TEE_TUIScreenOrientation

```
typedef enum
{
   TEE_TUI_PORTRAIT=0,
   TEE_TUI_LANDSCAPE
} TEE_TUIScreenOrientation;
```

The `TEE_TUIScreenOrientation` enumeration enumerates the supported screen orientations:

- `TEE_TUI_PORTRAIT` when the TUI screen is requested to be displayed as a portrait, i.e. vertically.
- `TEE_TUI_LANDSCAPE` when the TUI screen is requested to be displayed as a landscape, i.e. horizontally.

Values in the enumeration greater than or equal to `0x8000` are reserved for implementation-defined screen orientations.

4.4.4 TEE_TUIButtonType

```
typedef enum
{
   TEE_TUI_CORRECTION=0,
   TEE_TUI_OK,
   TEE_TUI_CANCEL,
   TEE_TUI_VALIDATE,
   TEE_TUI_PREVIOUS,
   TEE_TUI_NEXT
} TEE_TUIButtonType;
```

The `TEE_TUIButtonType` enumeration enumerates the six types of buttons that can be associated with TUI screens:

- `TEE_TUI_CORRECTION` represents the value for the CORRECTION button.
- `TEE_TUI_OK` represents the value for the OK button.
- `TEE_TUI_CANCEL` represents the value for the CANCEL button.
- `TEE_TUI_VALIDATE` represents the value for the VALIDATE button.
- `TEE_TUI_PREVIOUS` represents the value for the PREVIOUS button.
- `TEE_TUI_NEXT` represents the value for the NEXT button.

Values in the enumeration greater than or equal to `0x8000` are reserved for implementation-defined button types.

4.4.5 TEE_TUIImageSource

```
typedef enum
{
   TEE_TUI_NO_SOURCE=0,
   TEE_TUI_REF_SOURCE,
   TEE_TUI_OBJECT_SOURCE
} TEE_TUIImageSource;
```

The `TEE_TUIImageSource` enumeration enumerates the possible sources of an image:

- `TEE_TUI_NO_SOURCE` if no image is provided as input.
- `TEE_TUI_REF_SOURCE` if the image source is provided as a memory reference.
- `TEE_TUI_OBJECT_SOURCE` if the image source is provided as a Data Object in the Trusted Storage.

4.4.6 TEE_TUIImage

```
typedef struct
{
   TEE_TUIImageSource    source;
   union
   {
      struct
      {
         [inbuf] void* image; size_t imageLength;
      }
      ref;

      struct
      {
         uint32_t storageID;
         [in(objectIDLength)] void* objectID; size_t objectIDLen;
      }
      object;
   };
   uint32_t   width;
   uint32_t   height;
} TEE_TUIImage;
```

The `TEE_TUIImage` structure defines a way to handle an image for label area and buttons. An image source can be a buffer or an object in the Trusted Storage:

- `source` indicates the source of the image.
  - o If set to `TEE_TUI_NO_SOURCE` actually the structure does not refer to an image and all other fields are ignored.
  - o If set to `TEE_TUI_REF_SOURCE` the field `ref` MUST be selected.
  - o If set to `TEE_TUI_OBJECT_SOURCE` the field `object` MUST be selected.
- `image, imageLength` is a buffer containing the image. The referenced memory MUST contain the whole of the image in PNG format.
- `storageID` is the storage to use. It MUST be `TEE_STORAGE_PRIVATE` [TEE Internal API].

- `objectID, objectIDLen` is the Object Identifier which refers to a Data Object containing the image in PNG format.
- `width` represents the number of pixels of the width of the image.
- `height` represents the number of pixels of the height of the image.

The `width` and `height` MUST match the values set within the image in PNG format.

4.4.7 TEE_TUIScreenLabel

```
typedef struct
{
   char *         text;
   uint32_t       textXOffset;
   uint32_t       textYOffset;
   uint8_t        textColor[3];
   TEE_TUIImage   image;
   uint32_t       imageXOffset;
   uint32_t       imageYOffset;
} TEE_TUIScreenLabel;
```

The `TEE_TUIScreenLabel` structure defines the contents of the TA defined label area, which is provided to support TA branding and a TA defined message:

- `text` is the string to put in the label area. It can be `NULL`.
- `textXOffset` represents the x-coordinate for the top left corner of the text to render.
- `textYOffset` represents the y-coordinate for the top left corner of the text to render.
- `textColor` defines the color of the text in RGB form. Red is index 0, Green index 1, and Blue index 2. All components are specified in the range 0…255 but an implementation MAY re-interpret these values to suit its screen provided it makes the best possible match to the requested color.
- `image` is the image to be put in the label area.
- `imageXOffset` represents the x-coordinate for the top left corner of the image to display.
- `imageYOffset` represents the y-coordinate for the top left corner of the image to display.

For all coordinates described above, the coordinate origin is the top left corner of the label canvas.

4.4.8 TEE_TUIButton

The `TEE_TUIButton` structure defines the content of a button. The TA SHALL provide language or usage specific prompts or make use of the defaults. It is recommended that the defaults are graphical in nature to be as language agnostic as possible.

```
typedef struct
{
   char*          text;
   TEE_TUIImage   image;
} TEE_TUIButton;
```

- `text` is the string to associate with the button. It MAY be `NULL`.
- `image` is the image to associate with the button.

If an image is set as an input, its width and height MUST match the ones returned by `TEE_TUIGetScreenInfo`.

4.4.9 TEE_TUIScreenConfiguration

```
typedef struct
{
   TEE_TUIScreenOrientation screenOrientation;
   TEE_TUIScreenLabel    label;
   TEE_TUIButton*        buttons[TEE_TUI_NUMBER_BUTTON_TYPES];
   bool                  requestedButtons[TEE_TUI_NUMBER_BUTTON_TYPES];
} TEE_TUIScreenConfiguration;
```

The `TEE_TUIScreenConfiguration` structure enables configuration of a TUI screen:

- `screenOrientation` is the requested screen orientation.
- `label` specifies the label of the screen
- `buttons` customizes the buttons compared to the default configuration.
    - `buttons[TEE_TUI_CORRECTION]` is the CORRECTION button. If `NULL`, the default button configuration is used.
    - `buttons[TEE_TUI_OK]` is the OK button. If `NULL`, the default button configuration is used.
    - `buttons[TEE_TUI_CANCEL]` is the CANCEL button. If `NULL`, the default button configuration is used.
    - `buttons[TEE_TUI_VALIDATE]` is the VALIDATE button. If `NULL`, the default button configuration is used.
    - `buttons[TEE_TUI_PREVIOUS]` is the PREVIOUS button. If `NULL`, the default button configuration is used.
    - `buttons[TEE_TUI_NEXT]` is the NEXT button. If `NULL`, the default button configuration is used.
- `requestedButtons` specifies which buttons to be displayed. Each entry corresponds to a type of button and is set to `true` if the button is required to be displayed.
    - `requestedButtons[TEE_TUI_CORRECTION]` is related to the CORRECTION button.
    - `requestedButtons[TEE_TUI_OK]` is related to the OK button.
    - `requestedButtons[TEE_TUI_CANCEL]` is related to the CANCEL button.
    - `requestedButtons[TEE_TUI_VALIDATE]` is related to the VALIDATE button.
    - `requestedButtons[TEE_TUI_PREVIOUS]` is related to the PREVIOUS button.
    - `requestedButtons[TEE_TUI_NEXT]` is related to the NEXT button.

4.4.10 TEE_TUIScreenButtonInfo

```
typedef struct
{
   char*      buttonText;
   uint32_t   buttonWidth;
   uint32_t   buttonHeight;
   bool       buttonTextCustom;
   bool       buttonImageCustom;
} TEE_TUIScreenButtonInfo;
```

The `TEE_TUIScreenButtonInfo` structure represents button information associated with a TUI screen for a given orientation:

- `buttonText`: The value of the default label. `NULL` if not available.
- `buttonWidth`: The width in pixels of the button. MAY be `0` if the text and the image of the button cannot be customized.
- `buttonHeight`: The height in pixels of the button. MAY be `0` if the text and the image of the button cannot be customized.
- `buttonTextCustom`: `true` if the text of the button can be customized. `false` otherwise.
- `buttonImageCustom`: `true` if the image of the button can be customized. `false` otherwise.

Text and image cannot both be customized and `buttonTextCustom` and `buttonImageCustom` cannot both be set to `true`.

4.4.11 TEE_TUIScreenInfo

```
typedef struct
{
   uint32_t                  grayscaleBitsDepth;
   uint32_t                  redBitsDepth;
   uint32_t                  greenBitsDepth;
   uint32_t                  blueBitsDepth;
   uint32_t                  widthInch;
   uint32_t                  heightInch;
   uint32_t                  maxEntryFields;
   uint32_t                  entryFieldLabelWidth;
   uint32_t                  entryFieldLabelHeight;
   uint32_t                  maxEntryFieldLength;
   uint8_t                   labelColor[3];
   uint32_t                  labelWidth;
   uint32_t                  labelHeight;
   TEE_TUIScreenButtonInfo  buttonInfo[TEE_TUI_NUMBER_BUTTON_TYPES];
} TEE_TUIScreenInfo;
```

The `TEE_TUIScreenInfo` structure represents screen information for a given orientation:

- `grayscaleBitsDepth`: Available grayscale depth.
- `redBitsDepth`: Available Red bit depth.
- `greenBitsDepth`: Available Green bit depth.
- `blueBitsDepth`: Available Blue bit depth.
- `widthInch`: Width in pixels per inch.
- `heightInch`: Height in pixels per inch.
- `maxEntryFields`: Maximum number of entry fields that can be displayed on the screen. This is implementation dependent but the support of two entry fields at a minimum is mandatory.
- `entryFieldLabelWidth`: Width in pixels of the label of an entry field.
- `entryFieldLabelHeight`: Height in pixels of the label of an entry field.
- `maxEntryFieldLength`: The maximum number of characters that can be entered within an entry field.
- `labelColor`: The RGB values of the default label canvas. Red is index 0, Green index 1, and Blue index 2. All components are specified in the range 0…255 but an implementation MAY re-interpret these values to suit its screen provided it makes the best possible match to the requested color.
- `labelWidth`: Width in pixels of the label canvas.
- `labelHeight`: Height in pixels of the label canvas.
- `buttonInfo`: Information defining the buttons of the screens:
  - `buttonInfo[TEE_TUI_CORRECTION]` is the CORRECTION button.
  - `buttonInfo[TEE_TUI_OK]` is the OK button.
  - `buttonInfo[TEE_TUI_CANCEL]` is the CANCEL button.
  - `buttonInfo[TEE_TUI_VALIDATE]` is the VALIDATE button.
  - `buttonInfo[TEE_TUI_PREVIOUS]` is the PREVIOUS button.

- o `buttonInfo[TEE_TUI_NEXT]` is the NEXT button.

4.4.12 TEE_TUIEntryField

```
typedef struct
{
   char*                     label;
   TEE_TUIEntryFieldMode     mode;
   TEE_TUIEntryFieldType     type;
   uint32_t                  minExpectedLength;
   uint32_t                  maxExpectedLength;
   [outstring]               char* buffer; size_t bufferLength,
} TEE_TUIEntryField;
```

The `TEE_TUIEntryField` structure represents an entry field which acquires user inputs:

- `label`: The label associated with the entry field.
- `mode`: The mode to be used when displaying characters.
- `type`: The type of inputs accepted by the entry field.
- `minExpectedLength`: The minimum number of characters expected for the entry field. It is not possible to exit a TUI screen if the entry field does not contain this number of characters except to cancel it. If value is `0`, it is ignored.
- `maxExpectedLength`: The maximum number of characters expected for the entry field. Beyond this limit, characters entered by the user are ignored. If value is `0`, it is ignored. If value is different from `0`, it MUST be equal to or greater than `minExpectedLength`.
- `buffer, bufferLength`: Contains the input entered by the user. It MUST be big enough to contain characters specified by:
  - o `minExpectedLength` and `maxExpectedLength` when those fields are different from `0`
  - o `maxEntryFieldLength` of the `TEE_TUIScreenInfo` structure when `maxExpectedLength` is set to `0`

4.5 Functions

4.5.1 TEE_TUICheckTextFormat

```
TEE_Result TEE_TUICheckTextFormat(
[in]   char*      text,
[out]  uint32_t*  width,
[out]  uint32_t*  height,
[out]  uint32_t* lastIndex
)
```

Description

The `TEE_TUICheckTextFormat` function allows a TA to check whether a given text can be rendered by the current implementation and retrieves information about the size and width that is needed to render it.

`TEE_TUIInitSession` does not have to be called before using this function.

Parameters

- `text`: The string to be checked.
- `width`: Width in pixels needed to render the text.
- `height`: Height in pixels needed to render the text.
- `lastIndex`: Indicates the last character that has been checked. In case of success, it corresponds to the last character of the text string. In case of failure, it indicates the index of the character which causes the failure. The index starts at `0`.

Return Value

- `TEE_SUCCESS`: In case of success
- `TEE_ERROR_NOT_SUPPORTED`: If at least one of the characters present in the text string cannot be rendered.

Panic Reasons

- If any of the output parameters `width`, `height`, or `lastIndex` is `NULL` or points to an invalid region.

4.5.2 TEE_TUIGetScreenInfo

```
TEE_Result TEE_TUIGetScreenInfo(
[in]   TEE_TUIScreenOrientation    screenOrientation,
[in]   uint32_t                    nbEntryFields,
[out]  TEE_TUIScreenInfo*          screenInfo
)
```

Description

The `TEE_TUIGetScreenInfo` function retrieves information about the screen depending on its orientation and the number of required entry fields.

`TEE_TUIInitSession` does not have to be called before using this function.

Parameters

- `screenOrientation`: Defines for which orientation screen information is requested.
- `nbEntryFields`: Defines how many entry fields are requested.
- `screenInfo`: Returns information on the requested screen for a given orientation.

Return Value

- `TEE_SUCCESS`: In case of success
- `TEE_ERROR_NOT_SUPPORTED`: if the requested number of entry fields is not supported. In that case, the field `maxEntryFields` of the `screenInfo` output parameter is set to the maximum number of entry fields supported for the given orientation.

Panic Reasons

- If the parameter `screenInfo` is `NULL` or points to an invalid region.
- If the requested orientation is not supported as stated by the property `gpd.tee.tui.orientation`.

4.5.3 TEE_TUIInitSession

```
TEE_Result TEE_TUIInitSession(void)
```

Description

The `TEE_TUIInitSession` function claims an exclusive access to TUI resources for the current TA. Control of screen and keyboard MAY be taken over by the TEE at this stage. This just reserves the ability to use the TUI for this particular TA and will notify other TAs that this reservation has been made and the resource is busy (i.e. those other TAs will receive `TEE_ERROR_BUSY` when attempting this operation).

As a limited resource, the TUI session will be closed automatically whenever the TA does not display a TUI screen to interact with the user for a period of time. This period of time is equal to the value of the property `gpd.tee.tui.session.timeout`.

This function MUST be called before a screen can be displayed with `TEE_TUIDisplayScreen`.

Return Value

- `TEE_SUCCESS`: In case of success.
- `TEE_ERROR_BUSY`: If the TUI resources cannot be reserved.
- `TEE_ERROR_OUT_OF_MEMORY`: If the system ran out of resources.

4.5.4 TEE_TUICloseSession

```
TEE_Result TEE_TUICloseSession(void)
```

Description

The `TEE_TUICloseSession` function releases TUI resources previously acquired. This function SHOULD be called as soon as possible after the last TUI screen of the TUI session has ended in order to avoid a bad user experience.

Return Value

- `TEE_SUCCESS`: In case of success
- `TEE_ERROR_BAD_STATE`: If the current TA is not within a TUI session initially started by a successful call to `TEE_TUIInitSession`. In particular, it will be returned if a TUI session has been closed automatically because the TUI session timeout has been reached or if a TUI session has been closed due to an OS specific external event such as an incoming call as described in section 3.11.
- `TEE_ERROR_BUSY`: If the TUI resources are currently in use, i.e. a TUI screen is displayed. This error code can only be returned by a TEE implementation supporting multi-threading within a TA and will occur when a thread tries to close a TUI session that is displaying a TUI screen in another thread.

4.5.5 TEE_TUIDisplayScreen

```
TEE_Result TEE_TUIDisplayScreen(
[in]  TEE_TUIScreenConfiguration* screenConfiguration,
[in]  bool                         closeTUISession,
[in]  TEE_TUIEntryField*           entryFields,uint32_t entryFieldCount,
[out] TEE_TUIButtonType*          selectedButton
)
```

Description

The `TEE_TUIDisplayScreen` function displays a TUI screen. The display order of the requested entry fields is from top to bottom.

This function is cancellable, i.e., if the current task's cancelled flag is set and the TA has unmasked the effects of cancellation, then this function returns earlier than the requested timeout with the error code `TEE_ERROR_CANCEL`. See [TEE Internal API] section 4.10 for more details about cancellations.

In a given session, once the first call to `TEE_TUIDisplayScreen` has been made and if the parameter `closeTUISession` was set to `false`, the screen will not be handed back to the REE until `TEE_TUICloseSession` is called. When not displaying a particular `TEE_TUIDisplayScreen` the TEE MAY continue to display the current TUI screen or MAY display a screen indicating a security TUI session is in progress. Input events that occurred before the call to `TEE_TUIDisplayScreen` SHALL be ignored.

Parameters

- `screenConfiguration`: Configures the label of the screen and optionally the buttons of the screen.
- `closeTUISession`: If `true` the TUI session is automatically closed when exiting the function.
- `entryFields, entryFieldCount`: Array of entry fields. It contains the entry fields to display on the screen and enables input from the user by filling the `buffer` field of the corresponding `TEE_TUIEntryField` structure. It is ignored if `entryFieldCount` is set to `0`.
- `selectedButton`: In case of success, it indicates which button has been selected by the user to exit the TUI screen.

Note that all in and out parameters, as well as the buffers they refer to, MUST NOT reside in shared memory.

Return Value

- `TEE_SUCCESS`: In case of success.
- `TEE_ERROR_OUT_OF_MEMORY`: If the system ran out of resources.
- `TEE_ERROR_ITEM_NOT_FOUND`: If at least one image provided by the TA refers to a storage denoted by a `storageID` which does not exist or if the corresponding Object Identifier cannot be found in the storage.
- `TEE_ERROR_ACCESS_CONFLICT`: If at least one image provided by the TA refers to a Data Object in the Trusted Storage and an access right conflict was detected while opening the object.
- `TEE_ERROR_BAD_FORMAT`: If at least one input image is not compliant with PNG format.
- `TEE_ERROR_BAD_STATE`: If the current TA is not within a TUI session initially started by a successful call to `TEE_TUIInitSession`. In particular, it will be returned if a TUI session has been closed automatically because the TUI session timeout has been reached or if a TUI session has been closed due to an OS specific external event such as an incoming call as described in section 3.11.

- `TEE_ERROR_BUSY`: If the TUI resources are currently in use, i.e. a TUI screen is displayed. This error code can only be returned by a TEE implementation supporting multi-threading within a TA and will occur when a thread tries to display a TUI screen while a TUI screen is already displayed.
- `TEE_ERROR_CANCEL`: If the operation has been cancelled while a TUI screen is currently displayed.
  - The current UI session acquired by `TEE_TUIInitSession` is automatically closed as if `TEE_TUICloseSession` had been called.
  - The implementation MAY have started to fill out entry fields. In that case, entry fields will be returned with the last values entered by the user and it is up to the TA as to how it makes use of these.
- `TEE_ERROR_EXTERNAL_CANCEL`: If the operation has been cancelled by an external event which occurred in the REE while a TUI screen is currently displayed.
  - The current UI session acquired by `TEE_TUIInitSession` is automatically closed as if `TEE_TUICloseSession` had been called.
  - The implementation MAY have started to fill out entry fields. In that case, entry fields MAY be returned with the last values entered by the user and it is up to the TA as to how it makes use of these.

Panic Reasons

- If at least one of the in or out parameters, or one of the buffers they refer to, is in shared memory.
- If the parameter `screenConfiguration` is `NULL`.
- If the parameter `selectedButton` is `NULL`.
- If label fields do not match the values returned by the function `TEE_TUIGetScreenInfo` for the corresponding orientation and number of entry fields.
- If button fields do not match the values returned by the function `TEE_TUIGetScreenInfo` for the corresponding orientation and number of entry fields.
- If entry fields do not match the values returned by the function `TEE_TUIGetScreenInfo` for the corresponding orientation and number of entry fields. In particular if the length of at least one output string of an entry field does not follow the rules in section 4.4.12.
- If the requested buttons to be displayed do not match one of the authorized combinations described in section 3.3.

Annex A TUI API Usage

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

```
        TEE_TUIScreenInfo myScreenInfo;
        TEE_TUIScreenConfiguration myScreenConfig;
        TEE_TUIEntryField myEntryFields[2];
        char myLogin[26];
        char myPassword[26];
        TEE_TUIButtonType myKeyPressed;
        uint8_t * myLabelImage;
        size_t myLabelImageLength;
        uint32_t width, height, lastIndex;
        char myObjectID1[1] = {1};
        char myObjectID2[1] = {2};

        /* Check first if properties allow to apply a given TUI policy:
        - gpd.tee.tui.securityIndicator
        - gpd.tee.tui.languages
        - gpd.tee.tui.orientation
        - gpd.tee.tui.session.timeout (optionally)
        */

        /*************************************/
        /* Display of login/password screen */
        /*************************************/
        if ( TEE_TUIGetScreenInfo(TEE_TUI_PORTRAIT,2,&myScreenInfo) != TEE_SUCCESS )
        {
        /* The screen cannot be displayed. I probably input a non supported screen orientation ! */
        }

        /* Process label image: Access it and adapt it to offered possibilities (size, color)  */
        processMyLabelImage(myScreenInfo.grayscaleBitsDepth,
                            myScreenInfo.redBitsDepth,
                            myScreenInfo.greenBitsDepth,
                            myScreenInfo.widthInch,
                            myScreenInfo.heightInch,
                            myScreenInfo.labelColor,
                            myScreenInfo.labelHeight,
                            myScreenInfo.labelWidth,
                            &myLabelImage,
                            &myLabelImageLength);

        /* Adjust  text of buttons if possible/needed */
        if (myScreenInfo.buttonInfo[TEE_TUI_CORRECTION].buttonText != NULL)
        {
               /* Check it matches expectations (verify with a server ...) */
        }

        if (myScreenInfo.buttonInfo[TEE_TUI_VALIDATE].buttonText != NULL)
        {
               /* Check it matches expectations (verify with a server ...) */
        }

        if (myScreenInfo.buttonInfo[TEE_TUI_CANCEL].buttonText != NULL)
        {
               /* Check it matches expectations (verify with a server ...) */
        }

        /* Adjust screen label text */
        if (TEE_TUICheckTextFormat("logon the cloud",&width,&height,&lastIndex) != TEE_SUCCESS)
        {
               /*check which character was wrong with lastIndex.*/
        }

        if ( width > myScreenInfo.labelWidth || height > myScreenInfo.labelHeight )
        {
               /* Change the text to adapt */
        }

        /*Prepare the screen*/
        myScreenConfig.screenOrientation = TEE_TUI_PORTRAIT;
        myScreenConfig.label.textColor[0]=0;
        myScreenConfig.label.textColor[1]=0;
        myScreenConfig.label.textColor[2]=0;
        myScreenConfig.label.text = "logon the cloud";
        myScreenConfig.label.textXOffset = myScreenInfo.labelWidth - width;
        myScreenConfig.label.textYOffset = myScreenInfo.labelHeight / 2 - height;
        myScreenConfig.label.imageXOffset = 0;
        myScreenConfig.label.imageYOffset = 0;
        myScreenConfig.label.image.source = TEE_TUI_REF_SOURCE;
        myScreenConfig.label.image.ref.image = myLabelImage;
        myScreenConfig.label.image.ref.imageLength = myLabelImageLength;
        myScreenConfig.label.image.width = myScreenInfo.labelWidth;
        myScreenConfig.label.image.height = myScreenInfo.labelHeight;

        myScreenConfig.buttons[TEE_TUI_VALIDATE] = NULL;
        myScreenConfig.buttons[TEE_TUI_CANCEL] = NULL;
        myScreenConfig.buttons[TEE_TUI_CORRECTION] = NULL;
        myScreenConfig.buttons[TEE_TUI_NEXT] = NULL;
        myScreenConfig.buttons[TEE_TUI_PREVIOUS] = NULL;
        myScreenConfig.buttons[TEE_TUI_OK] = NULL;

        myScreenConfig.requestedButtons[TEE_TUI_VALIDATE] = true;
        myScreenConfig.requestedButtons[TEE_TUI_CANCEL] = true;
        myScreenConfig.requestedButtons[TEE_TUI_CORRECTION] = true;
        myScreenConfig.requestedButtons[TEE_TUI_NEXT] = false;
        myScreenConfig.requestedButtons[TEE_TUI_PREVIOUS] = false;
        myScreenConfig.requestedButtons[TEE_TUI_OK] = false;


        /* Adjust first entry field label text */
        if (TEE_TUICheckTextFormat("Please enter your login",
                                   &width,
                                   &height,
                                   &lastIndex) != TEE_SUCCESS)
        {
                /*check which character was wrong with lastIndex.*/
        }

        if ( width > myScreenInfo.entryFieldLabelWidth ||
             height > myScreenInfo.entryFieldLabelHeight )
        {
                /* Change the text to adapt */
        }

        /* Is the minimum length myScreenInfo.maxEntryFieldLength acceptable ? Then ... */
        if ( myScreenInfo.maxEntryFieldLength <25)
        {
                /* This is a problem  !! */
        }
        myEntryFields[0].type = TEE_TUI_ALPHANUMERICAL;
        myEntryFields[0].mode = TEE_TUI_CLEAR_MODE;
        myEntryFields[0].label = "Please enter your login";
        myEntryFields[0].minExpectedLength = 0;
        myEntryFields[0].maxExpectedLength = 25;
        myEntryFields[0].buffer = myLogin;
        myEntryFields[0].bufferLength = 26;

        /* Adjust second entry field label text */
        if (TEE_TUICheckTextFormat("Please enter your password",
                                  &width,
                                   &height,
                                   &lastIndex) != TEE_SUCCESS)
        {
                /*check which character was wrong with lastIndex.*/
        }

        if ( width > myScreenInfo.entryFieldLabelWidth ||
            height > myScreenInfo.entryFieldLabelHeight )
        {
                /* Change the text to adapt */
        }

        myEntryFields[1].type = TEE_TUI_ALPHANUMERICAL;
        myEntryFields[1].mode = TEE_TUI_HIDDEN_MODE;
        myEntryFields[1].label = "Please enter your password";
        myEntryFields[1].minExpectedLength = 0;
        myEntryFields[1].maxExpectedLength = 25;
        myEntryFields[1].buffer = myPassword;
        myEntryFields[1].bufferLength = 26;

        if ( TEE_TUIInitSession() != TEE_SUCCESS )
        {
                /* Denial of Service. Wait and retry...*/
        }

        switch(TEE_TUIDisplayScreen(&myScreenConfig,
                                    true,
                                    myEntryFields,
                                    2,
                                    &myKeyPressed))
        {
                case TEE_SUCCESS:
                        break;
                case TEE_ERROR_OUT_OF_MEMORY:
                case TEE_ERROR_BAD_STATE:
                case TEE_ERROR_BUSY:
                case TEE_ERROR_CANCEL:
                case TEE_ERROR_ACCESS_CONFLICT:
                case TEE_ERROR_ITEM_NOT_FOUND:
                case TEE_ERROR_EXTERNAL_CANCEL:
                        /* Handle error case */
                        break;
                default:
                        /* Instant trouble */
                        break;
        }

        if (myKeyPressed == TEE_TUI_VALIDATE)
        {
                /* Process login / password */
        }
        else
        {
                /* Operation has been cancelled by the user */
        }

        /*****************************************/
        /* Display two screens info to the user */
        /*****************************************/
        if ( TEE_TUIGetScreenInfo(TEE_TUI_PORTRAIT,0,&myScreenInfo) != TEE_SUCCESS )
        {
        /* The screen cannot be displayed. I probably input a non supported screen orientation ! */
        }

        /* Process label image for two screens */
        /* Actually, in that case, put them into files */
        processMyLabelImage2(myScreenInfo.grayscaleBitsDepth,
                             myScreenInfo.redBitsDepth,
                             myScreenInfo.greenBitsDepth,
                             myScreenInfo.widthInch,
                             myScreenInfo.heightInch,
                             myScreenInfo.labelColor,
                             myScreenInfo.labelHeight,
                             myScreenInfo.labelWidth);

        /* Adjust text of buttons if possible/needed */
        if (myScreenInfo.buttonInfo[TEE_TUI_NEXT].buttonText != NULL)
        {
                /* Check it matches expectations (verify with a server ...) */
        }

        if (myScreenInfo.buttonInfo[TEE_TUI_PREVIOUS].buttonText != NULL)
        {
                /* Check it matches expectations (verify with a server ...) */
        }

        if (myScreenInfo.buttonInfo[TEE_TUI_CANCEL].buttonText != NULL)
        {
                /* Check it matches expectations (verify with a server ...) */
        }

        if (myScreenInfo.buttonInfo[TEE_TUI_OK].buttonText != NULL)
        {
                /* Check it matches expectations (verify with a server ...) */
        }

        /*Prepare the First screen*/
        myScreenConfig.screenOrientation = TEE_TUI_PORTRAIT;
        myScreenConfig.label.text = NULL;
        myScreenConfig.label.imageXOffset = 0;
        myScreenConfig.label.imageYOffset = 0;
        myScreenConfig.label.image.source = TEE_TUI_OBJECT_SOURCE;
        myScreenConfig.label.image.object.storageID = TEE_STORAGE_PRIVATE;
        myScreenConfig.label.image.object.objectID = myObjectID1;
        myScreenConfig.label.image.object.objectIDLen = 1;
        myScreenConfig.label.image.width = myScreenInfo.labelWidth;
        myScreenConfig.label.image.height = myScreenInfo.labelHeight;

        myScreenConfig.buttons[TEE_TUI_VALIDATE] = NULL;
        myScreenConfig.buttons[TEE_TUI_CANCEL] = NULL;
        myScreenConfig.buttons[TEE_TUI_CORRECTION] = NULL;
        myScreenConfig.buttons[TEE_TUI_NEXT] = NULL;
        myScreenConfig.buttons[TEE_TUI_PREVIOUS] = NULL;
        myScreenConfig.buttons[TEE_TUI_OK] = NULL;

        myScreenConfig.requestedButtons[TEE_TUI_VALIDATE] = false;
        myScreenConfig.requestedButtons[TEE_TUI_CANCEL] = true;
        myScreenConfig.requestedButtons[TEE_TUI_CORRECTION] = false;
        myScreenConfig.requestedButtons[TEE_TUI_NEXT] = true;
        myScreenConfig.requestedButtons[TEE_TUI_PREVIOUS] = false;
        myScreenConfig.requestedButtons[TEE_TUI_OK] = false;

        if ( TEE_TUIInitSession() != TEE_SUCCESS )
        {
                /* DoS. Wait and retry...*/
        }

        switch(TEE_TUIDisplayScreen(&myScreenConfig,
                                    false,
                                    0,
                                    NULL,
                                    &myKeyPressed))
        {
                case TEE_SUCCESS:
                        break;
                case TEE_ERROR_OUT_OF_MEMORY:
                case TEE_ERROR_BAD_STATE:
                case TEE_ERROR_BUSY:
                case TEE_ERROR_CANCEL:
                case TEE_ERROR_ACCESS_CONFLICT:
                case TEE_ERROR_ITEM_NOT_FOUND:
                case TEE_ERROR_EXTERNAL_CANCEL:
                        /* Handle error case */
                        break;
                default:
                        /* Instant trouble */
                        break;
        }

        if (myKeyPressed != TEE_TUI_NEXT)
        {
                /* Operation has been cancelled by the user */
                TEE_TUICloseSession();
                /*exit ...*/

        }

        myScreenConfig.label.image.object.objectID = myObjectID2;
        myScreenConfig.label.image.object.objectIDLen = 1;

        myScreenConfig.requestedButtons[TEE_TUI_VALIDATE] = false;
        myScreenConfig.requestedButtons[TEE_TUI_CANCEL] = true;
        myScreenConfig.requestedButtons[TEE_TUI_CORRECTION] = false;
        myScreenConfig.requestedButtons[TEE_TUI_NEXT] = false;
        myScreenConfig.requestedButtons[TEE_TUI_PREVIOUS] = true;
        myScreenConfig.requestedButtons[TEE_TUI_OK] = true;

        switch(TEE_TUIDisplayScreen(&myScreenConfig,
                                    false,
                                    NULL,
                                    0,
                                    &myKeyPressed))
        {
                case TEE_SUCCESS:
                        break;
                case TEE_ERROR_OUT_OF_MEMORY:
                case TEE_ERROR_BAD_STATE:
                case TEE_ERROR_BUSY:
                case TEE_ERROR_CANCEL:
                case TEE_ERROR_ACCESS_CONFLICT:
                case TEE_ERROR_ITEM_NOT_FOUND:
                case TEE_ERROR_EXTERNAL_CANCEL:
                        /* Handle error case */
                        break;
                default:
                        /* Instant trouble */
                        break;
        }

        if (myKeyPressed == TEE_TUI_OK || myKeyPressed == TEE_TUI_CANCEL)
        {
                TEE_TUICloseSession();
        }
        else
        {
                /* go to previous screen */
        }

        /* ... */
```

Annex B Panicked Function Identification

If this specification is used in conjunction with the TEE Debug Specification [TEE Debug] then the specification number is 20 and the following values MUST be associated with the function declared.

Table B-1: Function Identification Values

Category	Function	Function Number in hexadecimal	Function Number in decimal
TUI implementation information			
	`TEE_TUICheckTextFormat`	0x0101	257
	`TEE_TUIGetScreenInfo`	0x0102	258
TUI session			
	`TEE_TUIInitSession`	0x0201	513
	`TEE_TUICloseSession`	0x0202	514
	`TEE_TUIDisplayScreen`	0x0203	515