

TCG TSS 2.0 TPM Command Transmission Interface (TCTI) API Specification

Family "2.0"

Version 1.0 Revision 18

24 January 2020

Contact: admin@trustedcomputinggroup.org

PUBLISHED

Copyright © TCG 2013-2020

TCG

Disclaimers, Notices, and License Terms

Copyright Licenses:

- Trusted Computing Group (TCG) grants to the user of the source code in this specification (the "Source Code") a worldwide, irrevocable, nonexclusive, royalty free, copyright license to reproduce, create derivative works, distribute, display and perform the Source Code and derivative works thereof, and to grant others the rights granted herein.
- The TCG grants to the user of the other parts of the specification (other than the Source Code) the rights to reproduce, distribute, display, and perform the specification solely for the purpose of developing products based on such documents.

Source Code Distribution Conditions:

- Redistributions of Source Code must retain the above copyright licenses, this list of conditions and the following disclaimers.
- Redistributions in binary form must reproduce the above copyright licenses, this list of conditions and the following disclaimers in the documentation and/or other materials provided with the distribution.

Disclaimers:

- THE COPYRIGHT LICENSES SET FORTH ABOVE DO NOT REPRESENT ANY FORM OF LICENSE OR WAIVER, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, WITH RESPECT TO PATENT RIGHTS HELD BY TCG MEMBERS (OR OTHER THIRD PARTIES) THAT MAY BE NECESSARY TO IMPLEMENT THIS SPECIFICATION OR OTHERWISE. Contact TCG Administration (admin@trustedcomputinggroup.org) for information on specification licensing rights available through TCG membership agreements.
- THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ACCURACY, COMPLETENESS, OR NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.
- Without limitation, TCG and its members and licensors disclaim all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

Any marks and brands contained herein are the property of their respective owners.

Corrections and Comments

Please send comments and corrections to admin@trustedcomputinggroup.org.

Normative-Informative Language

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this document’s normative statements are to be interpreted as described in RFC-2119, Key words for use in RFCs to Indicate Requirement Levels.

Revision History

Revision	Date	Description
Version 0.9 rev 00	05/30/17	<ul style="list-style-type: none"> First version of this document – created TCTI information removed from the SAPI specification.
Version 1.0 rev 01	06/02/17	<ul style="list-style-type: none"> Resolved review comments.
Version 1.0 rev 06	03/12/2018	<ul style="list-style-type: none"> Add description of standard initialization mechanisms, data structures and types.
Version 1.0 rev 07	03/13/2018	<ul style="list-style-type: none"> Resolve feedback from WG discussion.
Version 1.0 rev 08	03/15/2018	<ul style="list-style-type: none"> Reorder new initialization data types and structures in section 4.
Version 1.0 rev 09	03-15-2018	<ul style="list-style-type: none"> Replace “curly quotes” inserted by Microsoft Word with quote characters usable in C code.
Version 1.0 rev 10	03-16-2018	<ul style="list-style-type: none"> Fix formatting issue in section 4. Update date, revision number and admin contact email address on cover page
Version 1.0 rev 11	3-26-2018	<ul style="list-style-type: none"> Address TC review comments.
Version 1.0 rev 12	04-06-2018	<ul style="list-style-type: none"> Correct inconsistency in definition of TSS2_TCTI_INIT_FCN type between sections 3.4.1 and 4.2
Version 1.0 rev 18	16-12-2019	<ul style="list-style-type: none"> Resolved review comments.

Acknowledgements

TCG and the TSS Work Group would like to thank the following people for their work on this specification.

- Will Arthur Raytheon
- Brenda Baggaley Security Innovation (OnBoard Security)
- Dave Challenger Johns Hopkins University
- Mike Cox Security Innovation (OnBoard Security)
- Andreas Fuchs Fraunhofer SIT
- Ken Goldman IBM
- Jürgen Repp Fraunhofer SIT
- Philip Tricca Intel
- Lee Wilson Security Innovation (OnBoard Security)

Table of Contents

1	Definitions and References	7
1.1	Acronyms	7
1.2	TCG Software Stack 2.0 (TSS 2.0) Specification Library Structure	8
1.3	References.....	9
2	TCTI Introduction.....	10
2.1	TCTI Target Systems.....	10
3	TPM Command Transmission Interface.....	12
3.1	Introduction	12
3.1.1	Purpose & Goal	12
3.2	TCTI Context data structure	12
3.2.1	Magic & Version Fields.....	13
3.2.2	Function Invocation	13
3.2.3	Version Area.....	13
3.2.4	Non-opaque Area	14
3.2.5	Function Callbacks	14
3.2.6	Compatibility	18
3.2.7	Opaque Area	19
3.3	TCTI Info data structure	19
3.3.1	version	19
3.3.2	name.....	19
3.3.3	description	20
3.3.4	config_help	20
3.3.5	init	20
3.4	TCTI Initialization	20
3.4.1	Static or Dynamic Linking.....	20
3.4.2	Dynamic Loading.....	21
3.5	TCTI Loader Library.....	21
3.5.1	Tss2_TctiLdr_Initialize.....	21
3.5.2	Tss2_TctiLdr_Initialize_Ex	22
3.5.3	Tss2_TctiLdr_Finalize	23
3.5.4	Tss2_TctiLdr_GetInfo.....	23
3.5.5	Tss2_TctiLdr_FreeInfo	23
4	TCTI Header File.....	24
4.1	TCTI Prelude.....	24
4.2	TCTI Data Structures.....	24
4.3	TCTI Macros	25
4.4	Definitions Which Should Not Be Used by Callers	27
4.5	tss2_tcti.h Postlude.....	29
5	TCTI LDR Header File.....	30

TCG TSS System Level API and TCTI Specification

5.1	TCTI LDR prelude.....	30
5.2	Function Definitions	30
5.3	TCTI LDR postlude	30

1 Definitions and References

1.1 Acronyms

Term or Acronym	Definition
Application Binary Interface (ABI)	The ABI is the byte-wise layout of data types and function parameters in RAM as well as symbol definitions used to communicate between applications, shared objects and the kernel.
Application Programming Interface (API)	The API is the software interface defined by the functions and structures in a high-level programming language used to communicate between layers in the software stack.
Caller	The caller is the software that invokes a function call or that sends a TCTI command to the TAB/RM.
Connection	A “connection” to the TAB/RM corresponds to a TCTI context southbound from the SAPI to the TAB/RM.
ESAPI	TSS 2.0 Enhanced System API. This layer is intended to sit on top of the System API providing enhanced context management and cryptography.
FAPI	TSS 2.0 Feature API. This layer sits above the ESAPI and provides a high-level interface including a policy definition language and key store.
Implementation	An implementation is the source code and binary code that embodies a specification or parts of a specification.
Marshal	To marshal data is to convert data from C structures to marshaled data.
Marshaled Data	Marshaled data is the representation of data used to communicate with the TPM. In order to optimize data communication to and from the TPM, the smallest amount of data possible is sent to the TPM. For instance, if a structure starts with a size field and that field is set to 0, none of the other fields in the structure are sent to the TPM. Another example: if an input structure consists of a union of data structures, the marshaled representation will be the size of just the data structure selected from the union (actually the marshaled version of that structure itself). Also, the marshaled data must be in big-endian format since this is what the TPM expects.
NV	Non-volatile means that data is not lost when the system is powered down.
PCR	Platform Configuration Register (see TPM 2.0 Library Specification)
RM	The “Resource Manager” is software executing on a system with a TPM that ensures that the resources necessary to execute TPM commands are present in the TPM.
SAPI	TSS 2.0 System API. This layer is intended to sit on top of the TCTI providing marshaling/unmarshaling for TPM commands and responses.
TAB	The TPM Access Broker is software executing on a system with a TPM managing concurrent access from multiple applications.

TPM Command Transmission Interface (TCTI)	The TCTI is an IPC abstraction layer used to send commands to and receive responses from the TPM or the TAB/RM.
TPM	Trusted Platform Module
TPM Resource	Data managed by a TPM that can be referenced by a TPM handle. For TPM 2.0, this includes TPM objects (keys and data objects), TPM NV indices, TPM PCRs and TPM reserved handles and hierarchies.
TSS	TPM Software Stack
Unmarshal	To unmarshal data is to convert data from marshalled format to C structures.

1.2 TCG Software Stack 2.0 (TSS 2.0) Specification Library Structure

The documents that are part of the specification of the TSS 2.0 are:

- [1] TCG TSS 2.0 Overview and Common Structures Specification
- [2] TCG TSS 2.0 TPM Command Transmission Interface (TCTI) API Specification
- [3] TCG TSS 2.0 Marshaling/Unmarshaling (MU) API Specification
- [4] TCG TSS 2.0 System API (SAPI) Specification
- [5] TCG TSS 2.0 Enhanced System API (ESAPI) Specification
- [6] TCG TSS 2.0 Feature API (FAPI) Specification
- [7] TCG TSS 2.0 TAB and Resource Manager Specification

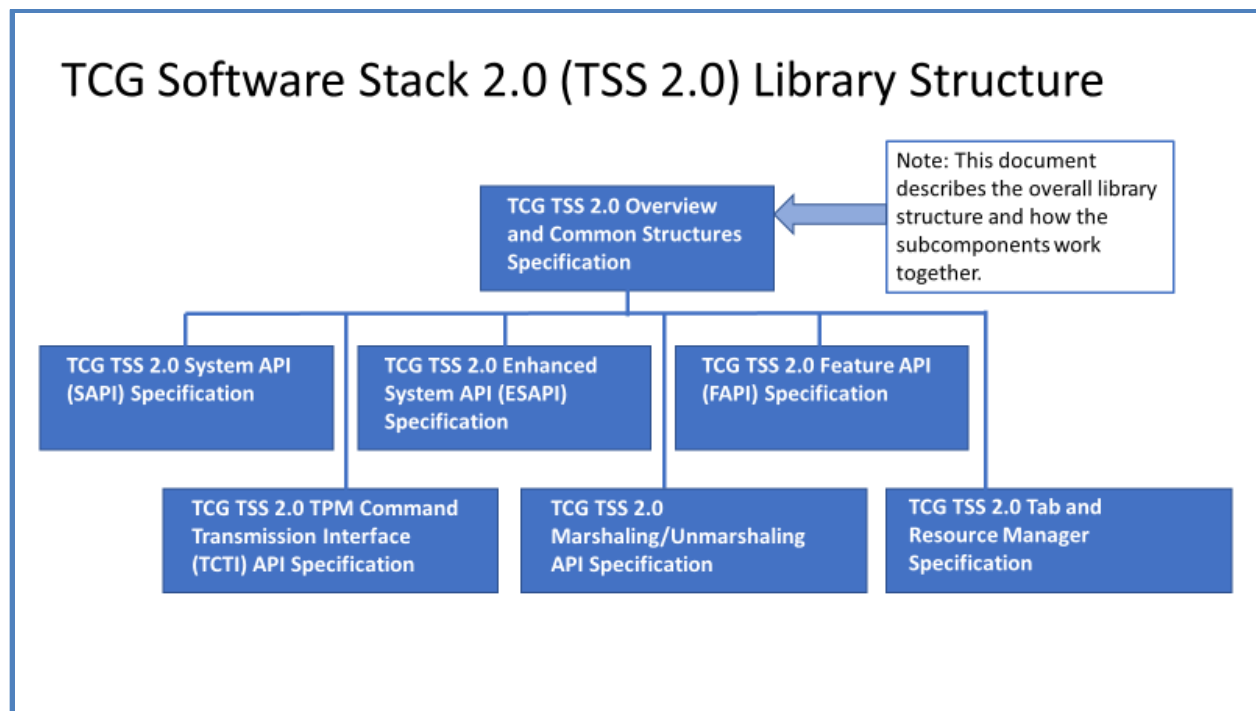


Figure 1: TSS 2.0 Specification Library

1.3 References

Documents change over time. The following rules apply to determining the edition of a reference needed for TSS 2.0. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

The following referenced documents are necessary or very useful for understanding the TPM 2.0 specification.

- [1] The Trusted Platform Module Library Specification, Family "2.0"
NOTE: More information, the specification, and other documents can be found at <https://trustedcomputinggroup.org/tpm-library-specification/> and <http://www.trustedcomputinggroup.org/work-groups/trusted-platform-module/>.

- [i] [Part 1: Architecture](#)
- [ii] [Part 2: Structures](#)
- [iii] [Part 3: Commands](#)
- [iv] [Part 3: Commands – Code](#)
- [v] [Part 4: Supporting Routines](#)
- [vi] [Part 4: Supporting Routines – Code](#)

2 TCTI Introduction

The TPM command transmission interface (TCTI) handles all the communication to and from the lower layers of the TSS software stack. For instance, different interfaces are required for local hardware TPMs, firmware TPMs, virtual TPMs, remote TPMs, and software TPM simulators.

NOTE: There are two different interfaces to TPMs: the legacy TIS interface and the command/response buffer (CRB).

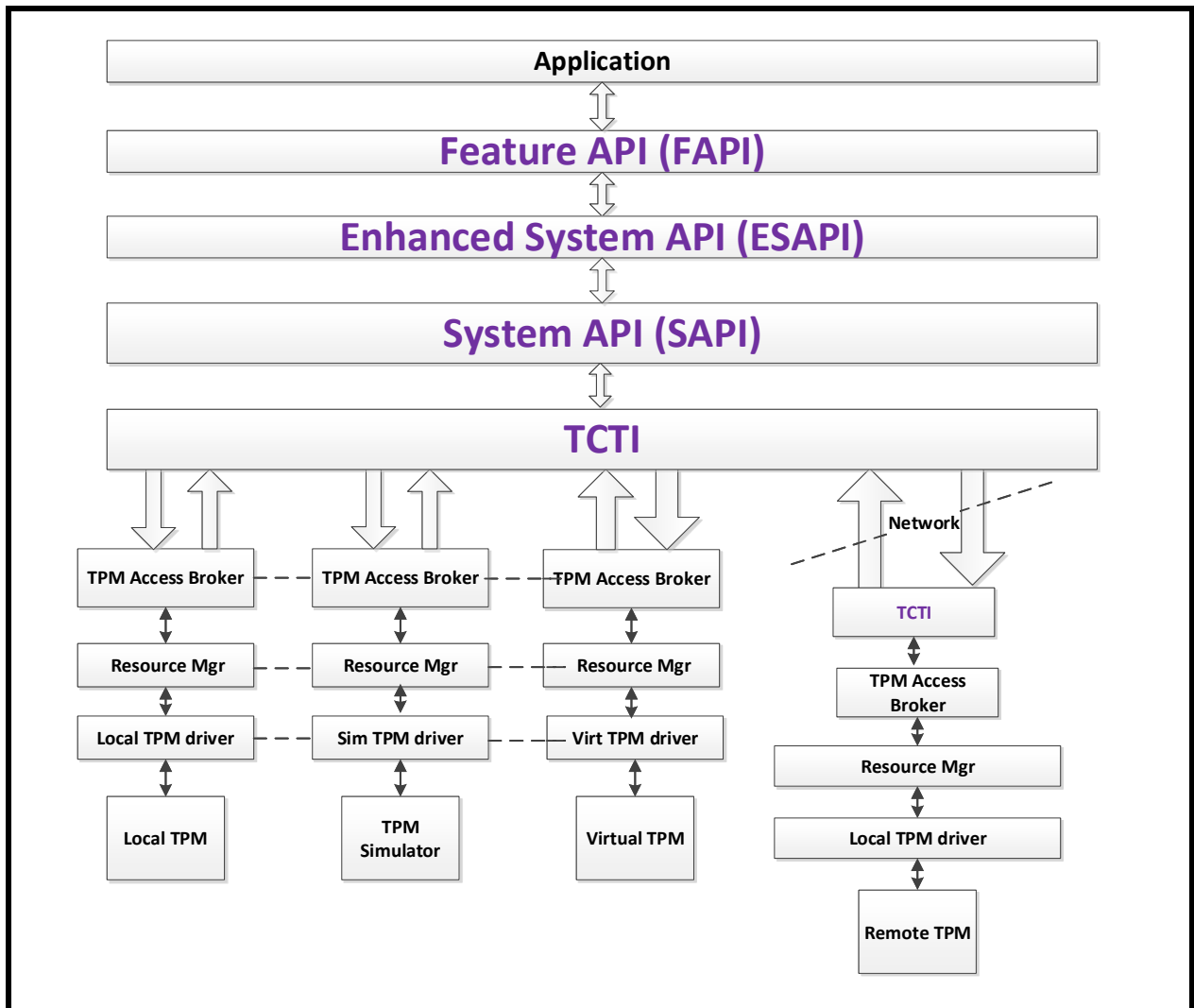


Figure 2: Using TCTI to Connect to Various Target TPMs

2.1 TCTI Target Systems

The TCTI API is designed to be used in a large range of computing devices from highly embedded systems to server OSes.

The TCTI is a low-level interface intended for expert applications. The SAPI uses the TCTI to communicate with the TPM. Use of the TCTI requires an understanding of common device driver interfaces. TCTI provides a generic interface to a wide variety of transport methods that can be used to communicate to the TPM.

3 TPM Command Transmission Interface

The TPM Command Transmission Interface (TCTI) is used to send marshalled commands to and receive marshalled responses from the TPM (or the underlying software stack that ultimately interacts with a TPM). It is designed to handle a wide variety of transmission methods.

3.1 Introduction

3.1.1 Purpose & Goal

The TPM Command Transmission Interface is designed to make it possible to switch modules at run time.

An application or the Feature-API may be configurable with regards to the “TCTI-drivers” they offer to a user. The TCTI is designed specifically for these use cases and provides conventions and helpers to be runtime-loadable friendly while still allowing compile-time linking without namespace clashes.

All TCTI data structures are included in the header file, `tss2_tcti.h`. The contents of `tss2_tcti.h` are specified in the “TSS 2.0 Header File Specification”.

3.2 TCTI Context data structure

The TCTI Context is an opaque pointer when passed in to the SAPI implementation. It is created by the caller, typically using a TCTI implementation. However, the SAPI implementation must extract several fields (typically function callbacks) in order to communicate with the next lower layer.

The SAPI implementation must not write any fields of the TCTI context.

The TCTI context consists of three parts.

- A version area
- A non-opaque area
- An opaque area

The structure definitions look generally like this. They are explained below.

```
typedef struct TSS2_TCTI_OPAQUE_CONTEXT_BLOB TSS2_TCTI_CONTEXT;

/* current version #1 known to this implementation */
typedef struct {
    uint64_t magic;
    uint32_t version;
    TSS2_RC (*transmit)( TSS2_TCTI_CONTEXT *tctiContext, size_t size,
const uint8_t *command);
    TSS2_RC (*receive) (TSS2_TCTI_CONTEXT *tctiContext, size_t *size,
uint8_t *response, int32_t timeout);
    void (*finalize) (TSS2_TCTI_CONTEXT *tctiContext);
    TSS2_RC (*cancel) (TSS2_TCTI_CONTEXT *tctiContext);
    TSS2_RC (*getPollHandles) (TSS2_TCTI_CONTEXT *tctiContext,
TSS2_TCTI_POLL_HANDLE *handles, size_t *num_handles);
    TSS2_RC (*setLocality) (TSS2_TCTI_CONTEXT *tctiContext, uint8_t locality);
} TSS2_TCTI_CONTEXT_COMMON_V1;

typedef struct {
```

```

    TSS2_TCTI_CONTEXT_COMMON_V1    v1;
    TSS2_TCTI_MAKE_STICKY_FCN      makeSticky;
} TSS2_TCTI_CONTEXT_COMMON_V2;

typedef TSS2_TCTI_CONTEXT_COMMON_V2 TSS2_TCTI_CONTEXT_COMMON_CURRENT;

```

3.2.1 Magic & Version Fields

The *magic* value is some unique number, perhaps simply a random number. The SAPI implementation will likely not read it. The value is used to perform a sanity check to ensure that a TCTI function is passed structures that it is capable of handling. Typically a TCTI module will only handle structures that it created.

The *version* value denotes the version number of the specification that defines a given structure. This value must monotonically increase from older to newer versions. The first version was 0x1. The current version is 0x2.

3.2.2 Function Invocation

In order to call any of these functions, an application first needs to check that a given TCTI Context has the correct version number, which is a version greater or equal to the version at the time that the function was added to the common function table. The application then needs to check whether the function is also implemented by the respective driver, that is, whether the function pointer is non-NULL. Only then can the function be safely called. This process can be encapsulated inside a helper macro such as the following example:

```

#define Tss2_Tcti_Transmit(tctiContext, size, command) \
    ((tctiContext == NULL) ? TSS2_TCTI_RC_BAD_REFERENCE: \
    (((TSS2_TCTI_CONTEXT_VERSION *)tctiContext)->version < 1) ? \
    TSS2_TCTI_RC_WRONG_ABI_VERSION : \
    (((TSS2_TCTI_CONTEXT_V1 *)tctiContext)->transmit == NULL) ? \
    TSS2_TCTI_RC_NOT_IMPLEMENTED: \
    ((TSS2_TCTI_CONTEXT_V1 *)tctiContext)->transmit(tctiContext, size, command))

```

A similar pattern can be implemented for all TCTI function pointers.

3.2.3 Version Area

The SAPI implementation extracts the version area by casting the opaque context to a TSS2_TCTI_CONTEXT structure. Each context structure must begin with the same members as the TSS2_TCTI_CONTEXT structure.

The SAPI implementation supports one or more versions of TCTI.

In the simplest case, the SAPI implementation supports one version. This would not necessarily be the latest version, but rather the numerically lowest version that contains all fields that the implementation requires. Using the lowest possible version permits the SAPI implementation to use the largest possible set of TCTI implementations.

In a more complex case, the SAPI implementation supports multiple TCTI versions. This might be useful if a version is deprecated, where a callback has been superseded by an improved function.

3.2.4 Non-opaque Area

The contents of a non-opaque area are defined by the *version*. The TCG defines versions and their contents. Each newer version must contain all fields of the previous version in the same order. Callbacks must have identical parameters and return values. This permits the SAPI implementation to safely cast a TCTI context to any version not greater than the value in the *version* field.

Once the context is cast, structure members can be dereferenced and used. A typical structure member is a function callback provided by the TCTI implementation.

3.2.5 Function Callbacks

The SAPI implementation extracts these callbacks from the non-opaque area of the TCTI context.

3.2.5.1 transmit

```
TSS2_RC (*transmit)(
    TSS2_TCTI_CONTEXT      *tctiContext,
    size_t                 size,
    const uint8_t          *command
);
```

This function transmits the command packet of size bytes to next layer below the caller.

Response Codes:

- TSS2_TCTI_RC_NOT_IMPLEMENTED: if the function isn't implemented

NOTE: this error code would only be used if future versions of this spec deprecated or removed the transmit call.

- TSS2_TCTI_RC_IO_ERROR: underlying IO failed
- TSS2_TCTI_RC_BAD_CONTEXT: bad version and/or magic fields in the TCTI-context
- TSS2_TCTI_RC_BAD_SEQUENCE: if transmit was called more than once without a call to receive in between
- TSS2_TCTI_RC_BAD_REFERENCE: if tctiContext or command is NULL
- TSS2_TCTI_RC_BAD_VALUE: if a bad value for any parameter is detected

NOTE: Some possible examples of bad values are: size out of range, size not at least 10 bytes, or size doesn't match commandSize.

3.2.5.2 receive

```
TSS2_RC (*receive)(
    TSS2_TCTI_CONTEXT      *tctiContext,
    size_t                 *size,
    uint8_t                *response,
    int32_t                timeout
);
```

This function receives a response packet from the layer below the caller.

If the parameter response is NULL and the size of the TPM response is known, the required size is written to the size parameter and TSS2_RC_SUCCESS is returned.

On input, *size* is the maximum allocated byte size of *response*. On a successful return, *size* is the actual used bytes of *response*. If *size* is insufficient for the TPM response the required size is returned in *size* with TSS2_TCTI_RC_INSUFFICIENT_BUFFER error.

NOTE: in some cases, depending on the communication interface, this will require the TCTI layer to keep track of the data read from the interface during the unsuccessful call to receive. A subsequent call to receive using the same TCTI context and a larger response buffer and *size* could be used to get the response data and clear the interface so that subsequent commands can be sent.

If *timeout* is TSS2_TCTI_TIMEOUT_BLOCK, the command is synchronous and blocks until a response is received.

If *timeout* is TSS2_TCTI_TIMEOUT_NONE, the command returns immediately. *size* and *response* are updated only on a successful return.

If *timeout* is positive, the command returns after a maximum of *timeout* msec. *size* and *response* are updated only on a successful return.

receive always returns either the entire TPM response or a return code indicating that the response is not yet completely available. A TCTI implementation that might receive partial responses can use the response buffer to assemble the partial responses.

There is one exception to the above: *receive* may return a return code indicating that the response buffer is too small to hold the TPM response. In that case, the caller must call *receive* again with a larger buffer.

Response Codes:

- TSS2_TCTI_RC_NOT_IMPLEMENTED: if the function isn't implemented

NOTE: this error code would only be used if future versions of this spec deprecated or removed the *receive* call.

- TSS2_TCTI_RC_BAD_CONTEXT: bad version and/or magic fields in the TCTI-context
- TSS2_TCTI_RC_TRY_AGAIN: if a timeout occurred before the complete response was received.
- TSS2_TCTI_RC_INSUFFICIENT_BUFFER: if the response buffer is too small for the TPM response. In this case, the returned *size* indicates the size of the buffer that is needed for the response.
- TSS2_TCTI_RC_IO_ERROR: Underlying IO failed.
- TSS2_TCTI_RC_BAD_REFERENCE: if *tctiContext* or *size* is NULL
- TSS2_TCTI_RC_BAD_VALUE: if *timeout* is negative but not -1 (TSS2_TCTI_TIMEOUT_BLOCK) or *response* is NULL.
- TSS2_TCTI_RC_BAD_SEQUENCE: if *receive* called again after first successful *receive* or if *transmit* not called first

3.2.5.3 finalize

```
void (*finalize) (
    TSS2_TCTI_CONTEXT          *tctiContext
);
```

This function performs any actions required when a TCTI connection is terminated and invalidates the TCTI Context. The TCTI Context cannot be used for subsequent operations after this call. This function should be called whenever a TCTI Context is not needed anymore. Afterwards the TCTI Context memory can be freed.

3.2.5.4 cancel

```
TSS2_RC (*cancel) (
    TSS2_TCTI_CONTEXT *tctiContext
);
```

This function causes the TCTI layer to cancel the command: in some TCTI implementations this may include sending the TPM cancel command. This command can only be called between transmit and receive calls.

NOTE: After calling cancel, receive still needs to be called.

Response Codes:

- TSS2_TCTI_RC_NOT_IMPLEMENTED: if the function isn't implemented
- TSS2_TCTI_RC_IO_ERROR: if connection to the TPM fails.
- TSS2_TCTI_RC_BAD_SEQUENCE: if not called between transmit and receive.
- TSS2_TCTI_RC_BAD_REFERENCE: if tctiContext is NULL
- TSS2_TCTI_RC_BAD_CONTEXT: bad version and/or magic fields in the TCTI-context

3.2.5.5 getPollHandles

```
TSS2_RC (*getPollHandles) (
    TSS2_TCTI_CONTEXT *tctiContext,
    TSS2_TCTI_POLL_HANDLE *handles,
    size_t *num_handles
);
```

NOTE: This function was added to support event-loop driven programming. If polling or select aren't needed, this function isn't necessary. Understanding this function requires a detailed understanding of poll and select calls.

This function retrieves the handles that can be used for polling or select. This function returns a set of handles that can be used to poll for incoming responses from the underlying software stack or TPM. The type for these handles is platform specific and defined separately in the declaration of TSS2_TCTI_POLL_HANDLE. In order to query the number of handles that a TCTI module needs to have monitored, the application may pass NULL for handles; in this case, it returns the number of handles. In pseudo-code this could be:

```
*getPollHandles (&tctiContext, NULL, &num);
TSS2_TCTI_POLL_HANDLES handles[num];
*getPollHandles (&tctiContext, &handles[0], &num);
poll(&handles[0], num, -1); // Platform specific syscall
```

Response Codes:

- TSS2_TCTI_RC_NOT_IMPLEMENTED: if the function isn't implemented
- TSS2_TCTI_RC_BAD_REFERENCE: if tctiContext, handles, or num_handles is NULL
- TSS2_TCTI_RC_INSUFFICIENT_BUFFER: if num_handles is too small
- TSS2_TCTI_RC_BAD_CONTEXT: bad version and/or magic fields in the TCTI-context

3.2.5.6 setLocality

```
TSS2_RC (*setLocality) (
    TSS2_TCTI_CONTEXT    *tctiContext,
    uint8_t              locality
);
```

This function sets the locality for the TPM.

The locality cannot be changed between transmit and receive

Response Codes:

- TSS2_TCTI_RC_NOT_IMPLEMENTED: if the function isn't implemented
- TSS2_TCTI_RC_BAD_REFERENCE: if tctiContext is NULL
- TSS2_TCTI_RC_IO_ERROR: if command fails due to an IO error
- TSS2_TCTI_RC_BAD_SEQUENCE: if locality is changed between transmit and receive calls
- TSS2_TCTI_BAD_VALUE: if TCTI can't support the locality, e.g. the locality doesn't even exist.
- TSS2_TCTI_RC_NOT_PERMITTED: if the change in locality is not permitted, e.g. the locality is supported by the TPM, but the lower layer doesn't allow changing the locality.
- TSS2_TCTI_NOT_SUPPORTED: if the lower layer doesn't support changing localities at all.
- TSS2_TCTI_RC_BAD_CONTEXT: bad version and/or magic fields in the TCTI-context

3.2.5.7 makeSticky

```
TSS2_RC (*makeSticky) (
    TSS2_TCTI_CONTEXT    *tctiContext,
    TPM_HANDLE           *handle,
    uint8_t              sticky
);
```

If *sticky* is 1, this function allows an application to request the underlying resource manager to make sure that a certain session, sequence or object is loaded in TPM memory. Making a handle sticky means that it will stay loaded in TPM memory, meaning the RM won't unload it. This allows another process (typically a kernel) to use the object without concern about whether it's loaded in TPM memory.

If *sticky* is 0, and *makeSticky* is called with the physical handle returned when it was made sticky, the object is made unsticky. Making a handle unsticky means that the RM is free to manage it as it desires.

If a session, sequence, or object is made sticky, it can only be made unsticky by the same tctiContext.

NOTE: If the finalize call to TCTI context is called, the associated connection breaks, or the application process exits, all sticky sessions, sequences, and objects will be flushed from the TPM.

This function is optional for TCTI and will only be implemented, i.e. non-NULL and not returning TSS2_TCTI_RC_NOT_IMPLEMENTED, if the TCTI layer is communicating with a resource manager.

Due to the nature of this function - being able to do a denial of service attack on the TPM - it should be restricted for usage by privileged applications and/or users only. Details on how to identify those or what privileged means are platform and stack specific and thereby out of scope for this specification.

The handle parameter is an input and output parameter. If *sticky* is 1, on input, *handle* is a virtual handle which is going to be made sticky; on output, *handle* is the real handle. If *sticky* is 0, on input, *handle* is a real handle which is going to be made unsticky; on output, *handle* is the virtual handle.

NOTE: In order to use “trusted key ring” from the Linux kernel, an application must be able to pass a real handle for a loaded object to the kernel. It can't pass a virtual handle because the kernel is bypassing the RM which means that no virtual to physical handle translation is possible.

When a session, sequence, or object is sticky, its previous virtual handle can't be used.

NOTE: for sessions this is required to avoid application/kernel conflicts because session state changes whenever it's used. For sequences and objects, this isn't strictly required, but it was specified this way for the sake of consistency.

Response Codes:

- TSS2_TCTI_RC_NOT_IMPLEMENTED: if the function isn't implemented
- TSS2_TCTI_RC_BAD_REFERENCE: if tctiContext or handle is NULL
- TSS2_TCTI_RC_IO_ERROR: if command fails due to an IO error
- TSS2_TCTI_RC_BAD_SEQUENCE: if function is called between transmit and receive calls
- TSS2_TCTI_RC_BAD_CONTEXT: bad version and/or magic fields in the TCTI-context
- TSS2_TCTI_RC_BAD_HANDLE: if the handle does not exist or is not owned by the TCTI context.
- TSS2_TCTI_RC_BAD_VALUE: if *sticky* is not 0 or 1.

3.2.6 Compatibility

This design anticipates that the non-opaque area might change over time, with new functions added and existing functions deprecated or even deleted (set to NULL). The section lists a few such use cases.

The SAPI implementation might receive an old but still completely useful context. Assuming it has a structure definition for this version (which is why it was recommended that the implementation use the oldest suitable structure), it casts to that version and continues.

3.2.6.1 Old and Not Useful Version

The SAPI implementation can receive an old TCTI context version that it cannot use because it requires a callback not available in the old version. The implementation detects the down level version (by detecting that it's less than required version) and returns an error.

The TCTI implementation must be updated to support the new requirements.

3.2.6.2 New but Useful Version

If the SAPI implementation receives a newer version than the one compiled in, it can cast the TCTI context to its down level version and use existing structure members. The cast is safe because new members are added at the end of the structure and existing members are never removed.

The implementation cannot access new members, but it was not coded to use them anyway.

3.2.6.3 New and Not Useful Version

The implementation might receive a newer context version where structure members that it requires have been superseded by newer members. For example, the TCTI implementation might provide a new callback with a different parameter list.

The TCTI implementation flags this situation by setting an older, now unimplemented function callback to NULL. After the cast, the SAPI implementation detects the NULL. It cannot continue because a function it requires is no longer available.

The SAPI implementation must be updated to use the new callback.

3.2.6.4 New Version with Deprecated Functions

If the SAPI implementation receives a context version with both deprecated and recommended function callbacks, it can handle it one of two ways.

If it was compiled to only handle a down level context, it will cast and use only the deprecated callbacks.

If it was compiled to handle several context versions, it can cast to a more recent version and use the recommended callbacks.

3.2.7 Opaque Area

The SAPI implementation cannot access the opaque area of the TCTI context. The TCTI implementation can add any implementation specific fields as needed to the opaque area.

The TCTI is expected to cast the context to its implementation specific context type based on the version number. Since the TCTI both creates and consumes the context, no incompatibility is expected.

The TCTI implementation may also validate that the magic value is as expected. This sanity check ensures that the pointer passed in is indeed a TCTI context known to the TCTI implementation.

3.3 TCTI Info data structure

```
typedef struct {
    uint32_t version;
    const char *name;
    const char *description;
    const char *config_help;
    TSS2_TCTI_INIT_FCN init;
} TSS2_TCTI_INFO;
```

Each field in the TCTI info structure provides information to the caller about the TCTI module and its initialization mechanism. The intended purpose of each field is described in the following subsections:

3.3.1 version

The 'version' field is used to hold the version value described in section **Error! Reference source not found.**

3.3.2 name

The 'name' field holds the unique name of the TCTI module and MUST NOT contain the ':' character. The TCTI implementation should assume that this string will be displayed to users.

3.3.3 description

The 'description' field is a human readable string describing the TCTI module. The TCTI implementation should assume that this string will be displayed to users as a way to communicate information about the TCTI.

3.3.4 config_help

The 'config_help' field is a human readable string describing the format of a configuration string that may be passed into the TCTI initialization function. The TCTI implementation should assume that this string will be displayed in informative messages to users.

3.3.5 init

The 'init' field holds a reference to an initialization function of type TSS2_TCTI_INIT_FCN. This reference is used to initialize a TCTI context structure.

3.4 TCTI Initialization

TCTI modules must be initialized by the caller. The initialization process consists of two steps: instantiating a TCTI context structure and initializing this structure such that it is usable by the caller. This specification defines two methods for initializing TCTI modules. The first is through the invocation of an initialization function with a standard prototype and TCTI specific name. This method is intended for software initializing a statically or dynamically linked TCTI module. The second method relies on platform specific dynamic loading mechanisms like dlopen on *nix OSs or LoadLibrary on Microsoft Windows.

3.4.1 Static or Dynamic Linking

Initialization of a TCTI context structure through static or dynamic linking is accomplished through a library specific initialization function. The type of the function is defined in tss2_tcti.h as:

```
typedef TSS2_RC (*TSS2_TCTI_INIT_FCN) (TSS2_TCTI_CONTEXT *tctiContext,
                                     size_t *size,
                                     const char *conf);
```

To prevent conflicts between symbols in applications that link against multiple TCTI libraries the symbol name SHOULD follow this naming convention:

```
Tss2_Tcti_<name>_Init
```

Libraries exporting an initialization function for static or dynamic linking MUST provide an implementation of the type TSS2_TCTI_INIT_FCN. The name of the symbol exporting this function MUST replace <name> with the name of their TCTI module. This string SHOULD be the same as the 'name' field in the TSS2_TCTI_INFO structure.

Initialization functions of this type MUST initialize the provided TCTI context 'tctiContext' of size 'size' according to the configuration string in 'conf'. The format of the configuration string is implementation specific. When provided with a NULL context this function MUST return the minimum size of the caller supplied TCTI context structure in the 'size' parameter. When provided with a NULL configuration string the TCTI module MUST initialize the provided 'context' with default values. These defaults are implementation specific. In the event that the caller provides a non-NULL context and a value in the size parameter that is inaccurate (a 'size' field larger than the buffer referenced in the 'context' field) then the behavior of the initialization function is undefined.

Response Codes:

- TSS2_TCTI_RC_BAD_VALUE is returned if any parameters contain unexpected values.
- TSS2_TCTI_RC_BAD_REFERENCE is returned if any parameters are NULL when they should not be.
- TSS2_TCTI_RC_BAD_CONTEXT is returned if the size of the TCTI context provided is insufficient.

3.4.2 Dynamic Loading

The static or dynamic linking initialization method requires that the application program know which TCTI module it is being linked against when the application is compiled. To support the dynamic loading of TCTI modules and to allow application code to dynamically load multiple TCTI's the library provides two mechanisms. The first mechanism described below provides a well-known symbol to expose information about the TCTI module, and the second is provided by TCTI Loader Library described in 3.5. The interface for the first mechanism is defined in `tss2_tcti.h` as:

```
typedef const TSS2_TCTI_INFO* (*TSS2_TCTI_INFO_FCN) (void);
```

TCTI libraries implementing the dynamic loading scheme MUST implement a function of type `TSS2_TCTI_INFO_FCN` and it MUST be exposed through the symbol `Tss2_Tcti_Info`. For convenience this string is provided in `tss2_tcti.h`:

```
#define TSS2_TCTI_INFO_SYMBOL "Tss2_Tcti_Info"
```

The `TSS2_TCTI_INFO_SYMBOL` constant is defined for use by application developers for discovering this function at runtime. Application writers are advised against linking to this symbol. `TSS2_TCTI_INFO_SYMBOL` symbol should only be used for dynamic loading ('`dlopen`' or equivalent).

Functions of type `TSS2_TCTI_INFO_FCN` take no parameters and MUST return a constant reference to a populated `TSS2_TCTI_INFO` structure. The pointer returned is a const reference and should never be modified or freed by the caller. Once the caller has obtained the `TSS2_TCTI_INFO` structure they may use the function pointer in the '`init`' field to initialize a TCTI context structure.

3.5 TCTI Loader Library

While the goal of the TCTI is to abstract away the details of the transport mechanism used to move command and response buffers between the application and the TPM, the application must still know which TCTI to use in a given context. Selection of a TCTI library in many cases will depend on the platform executing the application. Expecting all applications to implement a TCTI loading protocol as well as encoding the context required to select a TCTI appropriate for the platform / circumstance is not practical. The TCTI loader library is described here to assist applications in the management of the TCTI lifecycle.

3.5.1 Tss2_TctiLdr_Initialize

```
TSS2_RC Tss2_TctiLdr_Initialize (
    const char *nameConf,
    TSS2_TCTI_CONTEXT **tctiContext
);
```

The `Tss2_TctiLdr_Initialize` function allocates and initializes a TCTI context structure. The TCTI library used and its configuration are determined by the '`nameConf`' string parameter. This string is the concatenation of two substrings:

- 'name': a string identifying the TCTI library used to initialize the TCTI context
- 'conf': a TCTI specific string encoding configuration values for the TCTI. This string is constrained by the same requirements as the 'conf' parameter of the 'TSS2_TCTI_INIT_FCN' function as described in 3.4.1 and MUST be passed to the underlying TCTI unmodified.

When combined in the 'nameConf' parameter these two substrings MUST be separated by the ':' character (e.g. "name:conf").

If the caller provides a NULL 'nameConf' then `Tss2_TctiLdr_Initialize` MUST select a default TCTI for the caller and pass a NULL configuration string to this TCTI's initialization function. Individual components of the 'nameConf' string may be omitted by the caller in which case the `Tss2_TctiLdr_Initialize` MUST handle the following situations:

- If no ':' separator is present in 'nameConf' then `Tss2_TctiLdr_Initialize` MUST treat the string as the TCTI name. When instantiating this TCTI `Tss2_TctiLdr_Initialize` MUST provide a NULL string as the `conf` parameter.
- If the ':' separator is the first character in the `nameConf` string then `Tss2_TctiLdr_Initialize` MUST select a default TCTI module for the caller. The initialization function for this TCTI module MUST be passed the substring from `nameConf` following the separator as the `conf` parameter.
- If the ':' separator is the last character in the `nameConf` string then `Tss2_TctiLdr_Initialize` MUST pass the requested TCTI initialization function a NULL string as the `conf` parameter.

The memory required for the TCTI context is allocated by this function and returned to the caller through the 'tctiContext' parameter.

Response Codes:

This function MUST return all response codes generated by the underlying TCTI as part of the initialization process. Additionally it MUST return the following values:

- TSS2_TCTI_RC_MEMORY is returned if memory allocation fails
- TSS2_TCTI_RC_NOT_SUPPORTED is returned if the loader is unable to locate a TCTI library associated with 'name'
- TSS2_TCTI_RC_IO_ERROR is returned for failures in the underlying loading mechanism
- TSS2_TCTI_RC_BAD_REFERENCE is returned if the 'tctiContext' parameter is NULL

3.5.2 Tss2_TctiLdr_Initialize_Ex

```
TSS2_RC Tss2_TctiLdr_Initialize_Ex (
    const char *name,
    const char *conf,
    TSS2_TCTI_CONTEXT **tctiContext
);
```

The `Tss2_TctiLdr_Initialize_Ex` function allocates and initializes a TCTI context structure. The TCTI library used and its configuration are determined by the 'name' and 'conf' parameters. The 'name' parameter is a string identifying the TCTI library used to initialize the TCTI context. The mechanism used to locate and load this library is platform specific. If the provided 'name' is a NULL pointer then the TCTI loader library MUST select a default TCTI on behalf of the caller. The 'conf' parameter is constrained by the same requirements as the 'conf' parameter of the 'TSS2_TCTI_INIT_FCN' function as described in 3.4.1 and MUST be passed to the underlying TCTI initialization function unmodified.

The memory required for the TCTI context is allocated by this function and returned to the caller through the 'tctiContext' parameter.

Response Codes:

This function MUST return all response codes generated by the underlying TCTI as part of the initialization process. Additionally it MUST return the following values:

- TSS2_TCTI_RC_MEMORY is returned if memory allocation fails
- TSS2_TCTI_RC_NOT_SUPPORTED is returned if the loader is unable to locate a TCTI library associated with 'name'
- TSS2_TCTI_RC_IO_ERROR is returned for failures in the underlying loading mechanism
- TSS2_TCTI_RC_BAD_REFERENCE is returned if the 'tctiContext' parameter is NULL

3.5.3 Tss2_TctiLdr_Finalize

```
void Tss2_TctiLdr_Finalize (TSS2_TCTI_CONTEXT **ctx);
```

The `Tss2_TctiLdr_Finalize` function performs any necessary clean-up for the allocated TCTI context and any other implementation specific considerations. This includes the deallocation of all memory allocated by the `Tss2_TctiLdr_Initialize` function.

Response Codes: none

3.5.4 Tss2_TctiLdr_GetInfo

```
TSS2_RC Tss2_TctiLdr_GetInfo (const char *name, TSS2_TCTI_INFO **info);
```

The `Tss2_TctiLdr_GetInfo` function allows callers to query the TCTI loading mechanism for supported TCTIs. This function MUST allocate a `TSS2_TCTI_INFO` structure for the caller and it MUST be populated with data from the TCTI library associated with the 'name'. Data in the `TSS2_TCTI_INFO` structure that the underlying implementation is unable to populate MUST be set to NULL.

Response Codes:

- TSS2_TCTI_RC_MEMORY is returned if memory allocation fails
- TSS2_TCTI_RC_NOT_SUPPORTED is returned if the loader is unable to locate a TCTI library associated with 'name'
- TSS2_TCTI_RC_IO_ERROR is returned for failures in the underlying loading mechanism
- TSS2_TCTI_RC_BAD_REFERENCE is returned if the 'info' parameter is NULL

3.5.5 Tss2_TctiLdr_FreeInfo

```
void Tss2_TctiLdr_FreeInfo (TSS2_TCTI_INFO *info);
```

The `Tss2_TctiLdr_FreeInfo` function is intended to free all resources allocated by the `Tss2_TctiLdr_GetInfo` function. Its only parameter is a `TSS2_TCTI_INFO` instance previously allocated by `Tss2_TctiLdr_GetInfo`.

Response Codes: none

4 TCTI Header File

tss2_tcti.h

4.1 TCTI Prelude

```
#ifndef TSS2_TCTI_H
#define TSS2_TCTI_H

#include <stdint.h>
#include <stddef.h>
#include "tss2_common.h"
#include "tss2_tpm2_types.h"

#ifndef TSS2_API_VERSION_1_2_1_108
#error Version mismatch among TSS2 header files.
#endif
```

4.2 TCTI Data Structures

TSS2_TCTI_POLL_HANDLE: The TCTI supports an asynchronous mode of operation for processing TPM commands. After transmission of a command, the application can request to be notified when the response is available for reception. The TSS2_TCTI_POLL_HANDLE type is used as a data type for the handles that are used when querying for this notice. Since these handles are highly platform specific, they will change depending on the type of platform. For some platforms the handles are defined in the following; further handle types will be defined in future revisions of this specification. Note that not all TCTI-drivers have support for this function.

```
/*
 * "Public" TCTI definitions and operations.
 */

/* Define OS-specific TSS2_TCTI_POLL_HANDLE */
#ifdef WIN32
#include <windows.h>
typedef HANDLE TSS2_TCTI_POLL_HANDLE;
#elif defined(_POSIX_C_SOURCE)
#include <poll.h>
typedef struct pollfd TSS2_TCTI_POLL_HANDLE;
```



```

#else
typedef void TSS2_TCTI_POLL_HANDLE;
#ifndef TSS2_TCTI_SUPPRESS_POLL_WARNINGS
#pragma message "Info: Platform not supported for TCTI_POLL_HANDLES"
#endif /* TSS2_TCTI_SUPPRESS_POLL_WARNINGS */
#endif /* OS selection */

/* Constants used to control timeout behavior */
#define TSS2_TCTI_TIMEOUT_BLOCK -1
#define TSS2_TCTI_TIMEOUT_NONE 0

#define TSS2_TCTI_INFO_SYMBOL "Tss2_Tcti_Info"

/* TSS2_TCTI_CONTEXT is a data structure opaque to the caller. */
typedef struct TSS2_TCTI_OPAQUE_CONTEXT_BLOB TSS2_TCTI_CONTEXT;

/* Type of TCTI initialization function. */
typedef TSS2_RC (*TSS2_TCTI_INIT_FCN) (
    TSS2_TCTI_CONTEXT *tctiContext,
    size_t             *size,
    const char         *conf);

/* Descriptive data for a TCTI library and its init function */
typedef struct {
    uint32_t version;
    const char *name;
    const char *description;
    const char *conf_help;
    TSS2_TCTI_INIT_FCN init;
} TSS2_TCTI_INFO;

/* Function to expose TCTI library TSS2_TCTI_INFO structure */
typedef const TSS2_TCTI_INFO* (*TSS2_TCTI_INFO_FCN) (
    void);

```

4.3 TCTI Macros

The following macros simplify some basic TCTI tasks.

```

/* Macros to simplify invocation of functions from the common TCTI structure */
#define Tss2_Tcti_Transmit(tctiContext, size, command) \
    ((tctiContext == NULL) ? TSS2_TCTI_RC_BAD_CONTEXT: \
    (TSS2_TCTI_VERSION(tctiContext) < 1) ? \
    TSS2_TCTI_RC_ABI_MISMATCH: \
    (TSS2_TCTI_TRANSMIT(tctiContext) == NULL) ? \
    TSS2_TCTI_RC_NOT_IMPLEMENTED: \
    TSS2_TCTI_TRANSMIT(tctiContext)(tctiContext, size, command))
#define Tss2_Tcti_Receive(tctiContext, size, response, timeout) \
    ((tctiContext == NULL) ? TSS2_TCTI_RC_BAD_CONTEXT: \
    (TSS2_TCTI_VERSION(tctiContext) < 1) ? \
    TSS2_TCTI_RC_ABI_MISMATCH: \
    (TSS2_TCTI_RECEIVE(tctiContext) == NULL) ? \
    TSS2_TCTI_RC_NOT_IMPLEMENTED: \
    TSS2_TCTI_RECEIVE(tctiContext)(tctiContext, size, response, timeout))
#define Tss2_Tcti_Finalize(tctiContext) \
    do { \
        if ((tctiContext != NULL) && \
            (TSS2_TCTI_VERSION(tctiContext) >= 1) && \
            (TSS2_TCTI_FINALIZE(tctiContext) != NULL)) \
        { \
            TSS2_TCTI_FINALIZE(tctiContext)(tctiContext); \
        } \
    } while (0)
#define Tss2_Tcti_Cancel(tctiContext) \
    ((tctiContext == NULL) ? TSS2_TCTI_RC_BAD_CONTEXT: \
    (TSS2_TCTI_VERSION(tctiContext) < 1) ? \
    TSS2_TCTI_RC_ABI_MISMATCH: \
    (TSS2_TCTI_CANCEL(tctiContext) == NULL) ? \
    TSS2_TCTI_RC_NOT_IMPLEMENTED: \
    TSS2_TCTI_CANCEL(tctiContext)(tctiContext))
#define Tss2_Tcti_GetPollHandles(tctiContext, handles, num_handles) \
    ((tctiContext == NULL) ? TSS2_TCTI_RC_BAD_CONTEXT: \
    (TSS2_TCTI_VERSION(tctiContext) < 1) ? \
    TSS2_TCTI_RC_ABI_MISMATCH: \
    (TSS2_TCTI_GET_POLL_HANDLES(tctiContext) == NULL) ? \
    TSS2_TCTI_RC_NOT_IMPLEMENTED: \

```

```

    TSS2_TCTI_GET_POLL_HANDLES(tctiContext) (tctiContext,          handles,
num_handles))
#define Tss2_Tcti_SetLocality(tctiContext, locality)              \
    ((tctiContext == NULL) ? TSS2_TCTI_RC_BAD_CONTEXT:          \
    (TSS2_TCTI_VERSION(tctiContext) < 1) ?                      \
    TSS2_TCTI_RC_ABI_MISMATCH:                                  \
    (TSS2_TCTI_SET_LOCALITY(tctiContext) == NULL) ?            \
    TSS2_TCTI_RC_NOT_IMPLEMENTED:                              \
    TSS2_TCTI_SET_LOCALITY(tctiContext) (tctiContext, locality))
#define Tss2_Tcti_MakeSticky(tctiContext, handle, sticky)       \
    ((tctiContext == NULL) ? TSS2_TCTI_RC_BAD_CONTEXT:          \
    (TSS2_TCTI_VERSION(tctiContext) < 2) ?                      \
    TSS2_TCTI_RC_ABI_MISMATCH:                                  \
    (TSS2_TCTI_MAKE_STICKY(tctiContext) == NULL) ?            \
    TSS2_TCTI_RC_NOT_IMPLEMENTED:                              \
    TSS2_TCTI_MAKE_STICKY(tctiContext) (tctiContext, handle, sticky))

```

4.4 Definitions Which Should Not Be Used by Callers

All TCTI features can be accessed via the definitions above. It is strongly recommended that the following definitions not be used directly by callers.

```

/*
 * "Private" TCTI definitions.
 *
 * All TCTI features can be accessed via the definitions above. It is
 * strongly recommended that the following definitions not be used
 * directly by callers. These are made public in order to enable
 * implementation of the macros above.
 */

/* Function pointer types for the TCTI operations. */
typedef TSS2_RC (*TSS2_TCTI_TRANSMIT_FCN) (
    TSS2_TCTI_CONTEXT *tctiContext,
    size_t          size,
    uint8_t         const *command);
typedef TSS2_RC (*TSS2_TCTI_RECEIVE_FCN) (
    TSS2_TCTI_CONTEXT *tctiContext,

```

```

    size_t          *size,
    uint8_t         *response,
    int32_t         timeout);
typedef void (*TSS2_TCTI_FINALIZE_FCN) (
    TSS2_TCTI_CONTEXT *tctiContext);
typedef TSS2_RC (*TSS2_TCTI_CANCEL_FCN) (
    TSS2_TCTI_CONTEXT *tctiContext);
typedef TSS2_RC (*TSS2_TCTI_GET_POLL_HANDLES_FCN) (
    TSS2_TCTI_CONTEXT *tctiContext,
    TSS2_TCTI_POLL_HANDLE *handles,
    size_t *num_handles);
typedef TSS2_RC (*TSS2_TCTI_SET_LOCALITY_FCN) (
    TSS2_TCTI_CONTEXT *tctiContext,
    uint8_t locality);
typedef TSS2_RC (*TSS2_TCTI_MAKE_STICKY_FCN) (
    TSS2_TCTI_CONTEXT *tctiContext,
    TPM2_HANDLE *handle,
    uint8_t sticky);

typedef struct {
    uint64_t magic;
    uint32_t version;
    TSS2_TCTI_TRANSMIT_FCN transmit;
    TSS2_TCTI_RECEIVE_FCN receive;
    TSS2_TCTI_FINALIZE_FCN finalize;
    TSS2_TCTI_CANCEL_FCN cancel;
    TSS2_TCTI_GET_POLL_HANDLES_FCN getPollHandles;
    TSS2_TCTI_SET_LOCALITY_FCN setLocality;
} TSS2_TCTI_CONTEXT_COMMON_V1;

typedef struct {
    TSS2_TCTI_CONTEXT_COMMON_V1 v1;
    TSS2_TCTI_MAKE_STICKY_FCN makeSticky;
} TSS2_TCTI_CONTEXT_COMMON_V2;

typedef TSS2_TCTI_CONTEXT_COMMON_V2 TSS2_TCTI_CONTEXT_COMMON_CURRENT;

```

```

/* Macros to simplify access to values in common TCTI structure */
#define TSS2_TCTI_MAGIC(tctiContext) \
    ((TSS2_TCTI_CONTEXT_COMMON_V1*)tctiContext)->magic
#define TSS2_TCTI_VERSION(tctiContext) \
    ((TSS2_TCTI_CONTEXT_COMMON_V1*)tctiContext)->version
#define TSS2_TCTI_TRANSMIT(tctiContext) \
    ((TSS2_TCTI_CONTEXT_COMMON_V1*)tctiContext)->transmit
#define TSS2_TCTI_RECEIVE(tctiContext) \
    ((TSS2_TCTI_CONTEXT_COMMON_V1*)tctiContext)->receive
#define TSS2_TCTI_FINALIZE(tctiContext) \
    ((TSS2_TCTI_CONTEXT_COMMON_V1*)tctiContext)->finalize
#define TSS2_TCTI_CANCEL(tctiContext) \
    ((TSS2_TCTI_CONTEXT_COMMON_V1*)tctiContext)->cancel
#define TSS2_TCTI_GET_POLL_HANDLES(tctiContext) \
    ((TSS2_TCTI_CONTEXT_COMMON_V1*)tctiContext)->getPollHandles
#define TSS2_TCTI_SET_LOCALITY(tctiContext) \
    ((TSS2_TCTI_CONTEXT_COMMON_V1*)tctiContext)->setLocality
#define TSS2_TCTI_MAKE_STICKY(tctiContext) \
    ((TSS2_TCTI_CONTEXT_COMMON_V2*)tctiContext)->makeSticky

```

4.5 tss2_tcti.h Postlude

```

#endif /* TSS2_TCTI_H */

```

5 TCTI LDR Header File

tss2_tctildr.h

5.1 TCTI LDR prelude

```
#ifndef TSS2_TCTILDR_H
#define TSS2_TCTILDR_H

#include "tss2_tpm2_types.h"
#include "tss2_tcti.h"
```

5.2 Function Definitions

```
TSS2_RC Tss2_TctiLdr_Initialize (
    const char *nameConf,
    TSS2_TCTI_CONTEXT **tctiContext);
TSS2_RC Tss2_TctiLdr_Initialize_Ex (
    const char *name,
    const char *conf,
    TSS2_TCTI_CONTEXT **tctiContext);
TSS2_RC Tss2_TctiLdr_Finalize (
    TSS2_TCTI_CONTEXT **tctiContext);
TSS2_RC Tss2_TctiLdr_GetInfo (
    const char *name,
    TSS2_TCTI_INFO **info);
TSS2_RC Tss2_TctiLdr_FreeInfo (
    TSS2_TCTI_INFO **info);
```

5.3 TCTI LDR postlude

```
#endif /* TSS2_TCTILDR_H */
```