

TCG TSS 2.0 Response Code API Specification

Version 1.0

Revision 11

12/14/2018

admin@trustedcomputinggroup.org

PUBLIC REVIEW

Work in Progress

This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.

Disclaimers, Notices, and License Terms

THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Without limitation, TCG disclaims 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.

This document is copyrighted by Trusted Computing Group (TCG), and no license, express or implied, is granted herein other than as follows: You may not copy or reproduce the document or distribute it to others without written permission from TCG, except that you may freely do so for the purposes of (a) examining or implementing TCG specifications or (b) developing, testing, or promoting information technology standards and best practices, so long as you distribute the document with these disclaimers, notices, and license terms.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

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

DRAFT

Change History

REVISION	DATE	DESCRIPTION
Version 1.0 Revision 5	May 7, 2018	<ul style="list-style-type: none">Initial draft
Version 1.0 Revision 6	September 14, 2018	<ul style="list-style-type: none">Incorporate TC feedback
Version 1.0 Revision 10	November 6, 2018	<ul style="list-style-type: none">Included Scope in chapter 1. Handled final issues for balloting.
Version 1.0 Revision 11	December 3, 2018	<ul style="list-style-type: none">Incorporate TC feedback

DRAFT

Acknowledgements

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

- Brenda Baggaley, OnBoard Security
- Mike Cox, OnBoard Security
- Andreas Fuchs, Fraunhofer SIT
- Ken Goldman, IBM
- William Roberts, Intel
- Tadeusz Struk, Intel
- Philip Tricca, Intel
- Lee Wilson, OnBoard Security

DRAFT

Contents

Disclaimers, Notices, and License Terms.....	1
Change History.....	2
Acknowledgements	3
1 Information and Document Scope	6
1.1 Scope of this Specification	6
1.2 Acronyms	6
1.3 Key Words	6
1.4 TCG Software Stack 2.0 (TSS 2.0) Specification Library Structure.....	6
1.5 References.....	7
2 Introduction.....	8
3 Library Interface.....	9
3.1 Decoding Errors to a String.....	9
3.1.1 Error String Formats	9
3.2 Layer Handlers.....	10
3.2.1 Parameters	10
3.2.2 Return Value.....	11
4 Response Code Header File.....	12
4.1 tss2_rc.h prelude.....	12
4.2 tss2_rc.h type definitions.....	12
4.3 tss2_rc.h function prototypes	12
4.4 tss2_rc.h postlude.....	12

DRAFT

Table 1 – Default Layers 9

DRAFT

1 Information and Document Scope

1.1 Scope of this Specification

The TSS2 RC API defines an interface to a utility library. Its intended purpose is to provide application developers with a mechanism for converting instances of the TSS2_RC type into user-readable text. The intended audience for this specification includes software developers and designers implementing TCG TPM2 Software Stack APIs as well as those making use of TCG TPM2 Software Stack APIs in software systems.

1.2 Acronyms

For definitions of the acronyms used in the TSS 2.0 specifications please see the TCG TSS 2.0 Overview and Common Structures Specification [22].

1.3 Key Words

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

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

At the time of writing, the documents that specify 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 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
- [8] TCG TSS 2.0 TSS Response Code API Specification

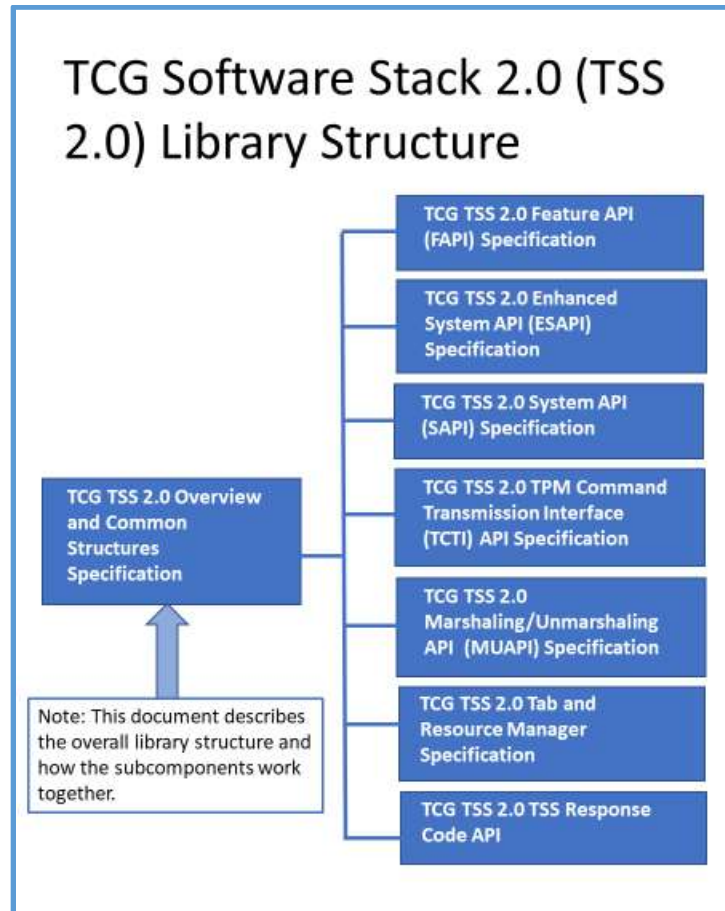


Figure 1: TSS 2.0 Specification Library

1.5 References

Most references for the TSS 2.0 specifications are provided in the TCG TSS 2.0 Overview and Common Structures Specification [1].

The following additional references are used by this specification:

- 1) IETF RFC 3447, Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
- 2) ISO C99 standard

2 Introduction

Response codes returned by TPM 2.0 devices and the TSS2 APIs follow the encoding scheme described in **Error! eference source not found.****Error! Reference source not found.**[5]. Developers using the TSS2 APIs need to provide actionable information to users of their applications (e.g. system administrators, end users). The `TSS2_RC` (`UINT32`) encoding is insufficient for this purpose as it leaves the burden of decoding this value into a human understandable form to the user.

This issue is addressed for c99 compliant systems through the `strerror` function[8]. The reader is assumed to be familiar with this function. The specification described in this document a similar method for mapping response code values in applications using the TSS2 API `TSS2_RC` type to human readable strings. Accommodations for the additional flexibility provided by the TSS2 layer encoding scheme in response codes are made through a callback registration mechanism described in section 3.3.

The reader is assumed to have an understanding of the response code encoding scheme documented in the section titled *Response Code Details* from [1], the section titled *TPM_RC (Response Codes)* from [2] and the section titled *Common Return Codes* from [5].

DRAFT

3 Library Interface

The following subsections describe the two functions in this library as well as specific requirements for implementers. The first function in this library maps `TSS2_RC` values returned from TSS2 API calls into human readable strings. The second is an interface to register functions for custom error decoding using the layer field described in [5].

3.1 Decoding Errors to a String

Function prototype:

```
const char *Tss2_RC_Decode(TSS2_RC rc);
```

The `'Tss2_RC_Decode'` function **MUST** map the provided `TSS2_RC` to a `NULL` terminated string. The contents of this string **MUST** adhere to the formats described in section 3.1.2. Implementations of this library **MUST** respect the `'const'` qualifier on values returned from this function and **MUST NOT** require the caller to perform memory management operations on the data returned.

NOTE: that passing the `TSS2_RC` value of `TSS2_RC_SUCCESS` results in the message of `'success'`.

3.1.1 Error String Formats

Strings returned by the `Tss2_RC_Decode` shall be in the format: `<layer-name>:<layer-specific-msg>`

Where the `<layer-name>` **MUST** be either a human readable string representation of the layer name, or the base 10 layer number if no handler is able to provide a name.

The `<layer-specific-msg>` **MUST** be a human readable string describing the response code, or the hex representation of the response code with the prefix `'0x'` when it cannot be decoded.

Example: assuming the default configuration and a non-decodable `TSS2_RC` value, if the `TSS2_RC` is of value `0x1002A` then the output would be: `1:0x2A`.

Table 1 – Default Layers

Layer ID	Layer Name	Description
0	tpm	Errors returned by the TPM itself.
6	fapi	Errors returned by the Feature API (FAPI).
7	esapi	Errors returned by the Enhanced System API (ESAPI).
8	sapi	Errors returned by the System API (SAPI).
9	mu	Errors returned by the marshaling library.
10	tcti	Errors returned by the Transmission Control Interface (TCTI).
11	rm	Errors returned by the Resource Manager (RM).
12	rmt	Errors returned by the Resource Manager (RM) TPM layer.

Implementations of this library **MUST** require no configuration before they are able to decode `TSS2_RC` values via calls to `Tss2_RC_Decode`. Additionally, they **MUST** be pre-configured to handle `TSS2_RC` values produced by the layers defined in Table 1. The decoding routine `Tss2_RC_Decode` **MUST** be limited to manipulating only thread local data. This allows different threads to have different layer handlers. A second call to this function from the same thread **MAY** invalidate data returned by previous calls.

3.1.1.1 TPM Layer 0 special messages

The *tpm* or layer 0 produces two distinct error message formats: format 0 and format 1.

3.1.1.1.1 Format 0 Messages

The *<layer-specific-msg>* field for format 0 response codes MUST be in the format: *<error|warn>(<1.2|2.0>):<description>*

Examples of Format 0 *<layer-specific-msg>* strings:

error(1.2): bad tag

warn(2.0): the 1st handle in the handle area references a transient object or session that is not loaded

Values for the *<description>* field are described in section 3.1.1.2.

3.1.1.1.2 Format 1 Messages

The *<layer-specific-msg>* field for format 1 response codes MUST be in the format: *<handle|session|parameter>(<index>):<description>* where the *index* value is displayed in its base 10 representation.

Values for the *<description>* field are described in section 3.1.1.2 below.

Examples of Format 1 Error Messages:

parameter(3):structure is the wrong size

3.1.1.2 Description Value

The text used in the *<description>* field MAY match the *description* field in the table titled *Definition of (UINT32) TPM_RC Constants (Actions) <OUT>* in [2]. If the implementation is unable to map the error field to a *description* string then the value from the error field MUST be displayed in hex notation with a leading '0x'.

3.2 Layer Handlers

The numeric layer values defined in [5] are for known layers as well as a range of values for use by libraries and applications. Implementations of these layers are provided with a collection of common error codes in [2]. To accommodate this flexibility, the response decoding library MUST provide an interface to register a layer specific response code decoding function. This function MUST have the following prototype:

```
TSS2_RC_HANDLER Tss2_RC_SetHandler(uint8_t layer, const char *name, TSS2_RC_HANDLER handler);
```

The `TSS2_RC_HANDLER` type MUST be defined in the `tss2_rc.h` header as:

```
typedef const char *(*TSS2_RC_HANDLER)(TSS2_RC rc);
```

3.2.1 Parameters

The *layer* parameter specifies the layer from which the *handler* is capable of decoding response codes. Valid values are in the range of 0 to 255.

The *name* parameter is a string used for the *<layer-name>* value in the format string described in section 3.1.1. The value MUST be either a string of up to 16 characters (not including the terminating null byte) or `NULL`. If `NULL` or length 0 (the empty string), this library MUST substitute a string representation of the layer number in base 10. If the string length is greater than 16, the string MUST be truncated to a length of 16.

The *handler* parameter specifies the callback function used to decode `TSS2_RC` values with the *layer* field set to the *layer* ID provided in the first parameter. Handler functions implementing this interface **MUST** rely only on the error bits in the `TSS2_RC` parameter as described in the section titled *Response Code Details* in [1]. Other fields in the `TSS2_RC` parameter **SHOULD** be set to 0 by the caller. If the handler function is passed a `TSS2_RC` value that it cannot decode then it **SHOULD** return `NULL`. If the `TSS2_RC` is known to the handler function, the value returned to the caller **MUST** be a static string representation of the error code. If the handler provided replaces a default layer, the descriptions returned by the new handler function **SHOULD** be formatted in the same way as the default handler function being replaced. If the *handler* parameter is `NULL` then the library **MUST** clear the currently registered handler function for the layer.

3.2.2 Return Value

`Tss2_RC_Set_Handler` **MUST** return the `TSS2_RC_HANDLER` replaced by the *handler* parameter. If no handler was previously registered for this layer then the value returned **MUST** be `NULL`.

DRAFT

4 Response Code Header File

tss2_rc.h

4.1 tss2_rc.h prelude

```
#ifndef TSS2_RC_H
#define TSS2_RC_H
```

```
#include <stdint.h>
#include <tss2/tss2_tpm2_types.h>
```

4.2 tss2_rc.h type definitions

```
typedef const char *(*TSS2_RC_HANDLER)(TSS2_RC rc);
```

4.3 tss2_rc.h function prototypes

```
const char *Tss2_RC_Decode(TSS2_RC rc);
TSS2_RC_HANDLER Tss2_RC_SetHandler(uint8_t layer, const char *name, TSS2_RC_HANDLER handler);
```

4.4 tss2_rc.h postlude

```
#endif
```

DRAFT