



# TEGRA BOARD SUPPORT PACKAGE FOR ANDROID

PR\_07297 | June 4, 2014 | Master Branch Release

**TLK Reference: FOSS Edition - Revision 1**



**Note:** This page intentionally left blank.

# Trusted Little Kernel

This Free Open Source Software (FOSS) release of the NVIDIA® Trusted Little Kernel (TLK) technology extends technology made available with the development of the Little Kernel (LK). You can download the LK modular embedded preemptive kernel for use on ARM, x86, and AVR32 systems at:

<https://github.com/travisg/lk>

LK features include the following support:

- Cortex-M3, Cortex-A8, AVR32, and x86 SoC families
- Multi-threading, IPCs, Ext2 file system, and thread scheduling

**Note:** LK does not contain any TrustZone Technology. For information about TrustZone modes and architecture, see the whitepaper at:

[http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C\\_trustzone\\_security\\_whitepaper.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf)

Earlier families of NVIDIA® Tegra® SoCs supported Trusted Foundations for security. Now NVIDIA implements its Trusted Little Kernel (TLK) technology, designed as a free, open-source trusted execution environment (OTE), for the following Tegra chipsets:

- Tegra K1 64 Bit families
- Tegra K1 32 Bit families
- Tegra® 4i families

Trusted Little Kernel (TLK) technology is not limited to the Tegra chipset.

TLK features include:

- Small, pre-emptive kernel
- Supports multi-threading, IPCs, and thread scheduling
- Added TrustZone features
- Added Secure Storage
- Under MIT/FreeBSD license

NVIDIA extensions to Little Kernel (LK) include:

- User mode
- Address-space separation for TAs
- TLK Client Application (CA) library
- TLK TA library
- Adding user space (Bionic)
- Crypto library (encrypt/decrypt, key handling) via Open SSL
- Linux kernel driver (tlk\_driver)
- Cortex A9/A15 support
- Power Management--Deep Sleep (LP0), Suspend (LP1), Standby (LP2)
- TrustZone memory carve-out (reconfigurable)
- Page table management
- On-the-fly (OTF) decoding driver for one-shot HW decode/encrypt
- Debugging support over UART (USB planned)

## Overview

The Android OS and NVIDIA® Trusted Little Kernel (TLK) software operate in a master-slave relationship; with TLK as the slave. TLK is compiled, using the GCC, when building your NVIDIA® Tegra® BSP for Android release. To facilitate debugging, TLK resides in a separate partition. The TLK current binary size is ~ 670 KB (with Widevine, HDCP, etc.) with 2 MB of reconfigurable carve-out.

NVIDIA TLK is comprised of the following distinct environments and components:

- **Non-Secure Environment (NSE):** Operating system (OS) for the device running in normal mode.
- **Open Trusted Environment (OTE):** A separate, software-partitioned, environment that provides trusted operations.

For example, OTE verifies that an application running in the OTE is not altered from its original, trusted condition.

- **Monitor:** Handles all communication between the NSE and OTE.
  - Activates the appropriate environment
  - Deactivates the appropriate environment
  - Routes function calls to the appropriate TLK libraries
  - Performs all environment switching

Switching occurs at the hardware level and is immediate. All applications running in the deactivated environment get suspended immediately.

- **Secure storage:** consists of several components that work together to store and access secured and encrypted data in a folder.
  - Components run inside and outside the OTE
  - Stored data can be generated by TLK itself or by a secure TA
  - Secure storage components manage security and encryption.

The folder where the secure storage components save data provides a large storage area and is not restricted by the size of the TLK partition.

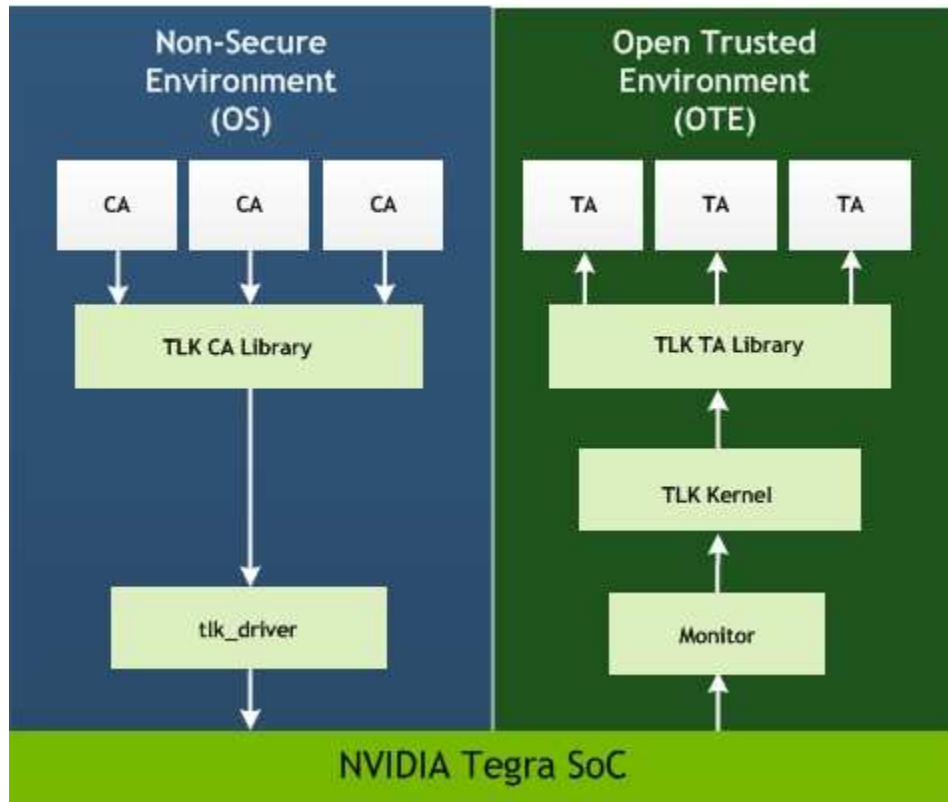
Additionally, the folder backs up the partition. The /data partition gets deleted upon factory reset and when obtaining root access with the `fastboot oem unlock` command.

For more information, see [Secure Storage](#) in this chapter.

To use the OTE and secure storage, every **client application** (CA) running in the NSE (non-secure environment) **must** have a partner **trusted application** (TA) running in the secure world (OTE). The secure TA is the only component that can communicate with the secure storage service, as the secure storage service only accepts requests originating from a secure TA.

- All secure operations are initiated by a CA running in the NSE.
- A TA in the OTE never initiates contact with the NSE.

The following figure shows the relationships among the components.



Trusted Applications (TA) are post-linked at compile time, and TA properties are determined using the manifest. TAs have their own sandbox, and TA-to-TA communication is supported. Drivers are also considered TAs.

## Execution Steps

1. When a client application (CA) needs a secure operation to be performed, it sends a request to a trusted application (TA) by invoking functions in the TLK CA library.
2. The TLK Driver routes the CA request to monitor, which routes the request through the TLK Kernel to the TLK TA Library.
3. The TLK TA library determines the appropriate TA to handle the request.
4. If the appropriate TA is not already running, the TLK TA library starts one.
5. The TLK TA library then passes control to the TA to handle the request.
6. Upon completion, execution control returns along the same path until reaching the CA, which receives a return value and any processed data.

Only one execution environment is active at any point in time. When the NSE is active, CAs can execute and the Monitor suspends TAs. Conversely, when the

Open Trusted Environment (OTE) is active, TAs can execute while the CAs are suspended.

## TLK Libraries

The functions contained in the TLK CA and TLK TA libraries operate synchronously. Calls to these functions do not return until the requested operation has completed. You can find the TLK libraries at the following location:

```
$TOP/3rdparty/ote_partner/lib
```

Note:

There are multiple libraries that are linked to the TLK TA libraries. These include:

- TLK Interface library—a common interface for command execution.
- Bionic—the version found in the Android AOSP
- OpenSSL—the version found in the Android AOSP

The following table describes the TLK directories.

Directory	Description
3rdparty/ote_partner/tlk	TLK core source
3rdparty/ote_partner/lib	Library interfaces/client (API's) libs
3rdparty/ote_partner/tlk_driver	Linux driver between the NSE and Secure worlds. This source code is provided as an example only.
3rdparty/ote_partner/daemon	A proxy agent in the NSE world for TLK This source code is provided as an example only.
3rdparty/ote_partner/tasks	A secure task (TA).
3rdparty/ote_partner/tools	Toolchain to build TLK. You must independently obtain these tools. For more information, see <a href="#">Obtaining the Toolchain</a> in this chapter.

## Secure Boot

NVIDIA® Trusted Little Kernel (TLK) software boots in the secure boot sequence.

### BootROM:

1. Verifies the BCT and boot loader using PKC boot.
2. Jumps to the boot loader.

**Boot loader:**

3. Copies the Encrypted Key Storage (EKS) partition to TZ-SRAM.
4. Copies the TF\_SBK to TZ-SRAM.
5. Verifies the TLK partition by using the PKC public key.
6. Jumps to the TLK partition with parameters such as the kernel boot address and TZ-DRAM parameters.

**TLK:**

7. Initializes the Monitor and sets up TZ-DRAM memory carve-out.
8. Copies the EKS partition from TZ-SRAM to TZ-DRAM.
9. Copies the TF\_SBK to TZ-DRAM.
10. Sets the mode to non-secure and jumps to the kernel.

**Linux kernel:**

11. Boots Android.

## Secure Storage

NVIDIA® Trusted Little Kernel (TLK) technology uses the `/data/tlk` folder from the Android File system as the top-level secure storage directory. Secure storage files are placed in subdirectories under `/data/tlk` that use the secure storage client's UUID for their name. TLK encrypts and then stores secure data in files in these subdirectories where it can be subsequently decrypted and accessed.

A daemon running in non-secure space provides `/data/tlk` access to TLK. The TLK daemon performs the following basic file operations originating from the OTE:

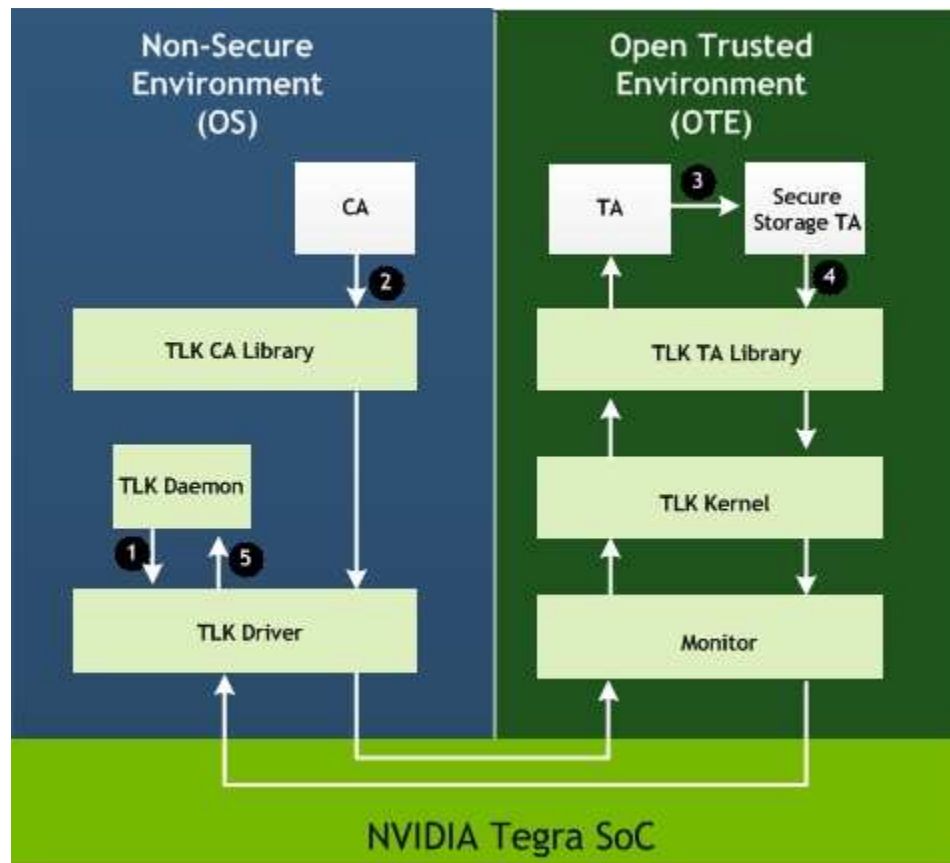
- create/delete
- open/close
- read/write
- getsize
- seek
- truncate

You can find the source code for the daemon here:

```
$TOP/3rdparty/ote_partner/daemon
```



## Secure storage flow



The following table describes the numbered interactions in the diagram.

Interaction	Description
1	The TLK daemon is blocked on an IOCTL call waiting to service the next secure storage request.
2	A CA makes a secure storage request to a TA.
3	The TA sends the request to the Secure Storage TA.
4	The Secure Storage TA processes the request and forwards a packet to the TLK driver in the non-secure world.
5	The TLK driver notifies the TLK daemon that a new request is ready. The TLK daemon fetches the new request and issues the corresponding file operations and notifies the TLK driver that the request is complete.

The TLK daemon loads via the following initialization routine:

```
$TOP/device/nvidia/common/init.tlk.rc
```

The TLK daemon runs as user `system` and group `keystore`. During boot, the daemon starts with an input parameter of `--storagedir <dirname>`, which

tells it which directory to use. This is a required parameter. Without it, the daemon will not boot.

The TLK daemon links with the `libtlk_client`, which provides APIs used to communicate with the Linux kernel. These APIs internally talk with the kernel via IOCTLs.

After boot, the TLK daemon makes an IOCTL call to the kernel to determine whether or not there is a new file operation request. If none, then the kernel blocks this IOCTL call until it has a new request.

Upon receiving a request from the TLK kernel, the TLK driver running in the Linux kernel packages the request for the TLK daemon and unblocks the daemon from inside the IOCTL call. Upon returning, the daemon parses the packet to determine and execute the corresponding file operations.

### **Create Operation**

In create operations the daemon creates a file using the filename specified in the request. A file with that name must not already exist. The file is created under `/data/tlk/<uuid>`, where `<uuid>` is a text representation of the client TA's UUID.

### **Delete Operation**

In delete operations the daemon deletes the file specified in the request.

### **Open Operation**

In open operations the daemon opens the file specified in the request and returns a handle that can be used for subsequent read/write/seek/getsize/truncate operations. The open operation parameters include access flags for specifying read, write, or read-write access to the file.

### **Close Operation**

In close operations the daemon closes access to the file using the handle from a previous open operation.

### **Read Operation**

In read operations the daemon reads the specified number of bytes at the current file position using the specified handle. The handle must be generated in a

previous open operation. The data is copied to the specified buffer for return to requester. The actual number of bytes read is returned to the caller.

### **Write Operation**

In write operations the daemon writes the specified number of bytes at the current file position using the specified handle. The handle must be generated in a previous open operation. The data written is copied from the specified buffer.

### **Get Size Operation**

In getsizes operations the daemon returns the current size in bytes of the file represented by the specified handle. The handle must be generated in a previous open operation.

### **Seek Operation**

For seek operations the daemon sets the current position in the file represented by the specified handle to the specified offset. The handle must have been generated from a previous open operation.

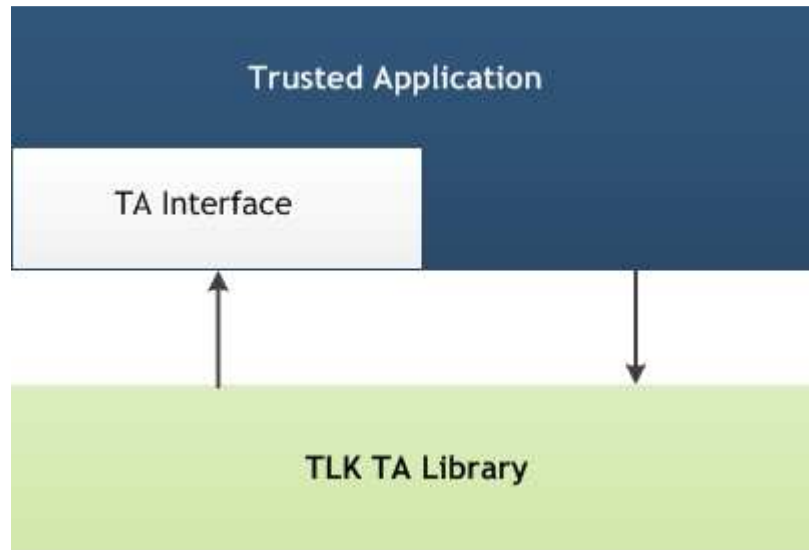
### **Truncate Operation**

In truncate operations the daemon truncates the file represented by the handle to the specified length. The handle must be generated in a previous open operation. The specified length must be less than or equal the current size of the file.

## **TLK Trusted Application Development**

A trusted application (TA) performs secure operations requested by a client application (CA). The CA request is routed to the TLK TA Library which calls the TA through a common function interface. This common TA interface is a group of functions that every TA must implement. The TLK TA Library also provides functions that a TA may call to perform data manipulation, cryptography, and other trusted operations.

The following diagram illustrates the relationships between a TA, its interface, and the TLK TA Library.



## Building a TLK Device

This section describes how to build an Android-based TLK device.

### Required Components

- `tlk`—the secure Trusted Little Kernel (TLK).
- `tasks`—the secure tasks that run under the TLK.
- `lib`—the library support for tasks.
- `client`—library support for writing client applications that run under Linux in the non-secure-world. It also contains some example clients and tasks.

## Obtaining the Toolchain

Before you can build TLK, you must get the toolchain and populate the following directory:

```
3rdparty/ote_partner/tools
```

The required toolchain depend on your architecture:

- For 64-bit TLK: `tools/aarch64-linux-android-4.8`
- For 32-bit TLK: `tools/arm-eabi-4.7`

You can obtain these toolchains from:

<https://android.googlesource.com/platform/prebuilts/gcc/linux-x86/aarch64/aarch64-linux-android-4.8>  
<https://android.googlesource.com/platform/prebuilts/gcc/linux-x86/arm/arm-eabi-4.7>

## Building a Task

A task is compiled as an ARM Elf Executable using a custom linker script. Below is the `Android.mk` for the Storage TA sample `test_app`.

### Example Android Makefile

```
ifeq (tlk,$(SECURE_OS_BUILD))

LOCAL_PATH:= $(call my-dir)

## Make internal task
include $(NVIDIA_DEFAULTS)

LOCAL_MODULE_TAGS := optional
LOCAL_C_INCLUDES := $(LOCAL_PATH) \
                    $(TOP)/3rdparty/ote_partner/lib/include
LOCAL_SRC_FILES := storagedemo.c manifest.c
LOCAL_MODULE := tlkstoragedemo_task
LOCAL_STATIC_LIBRARIES:= \
    libtlk_service \
    libtlk_common \
    libc

LOCAL_LDFLAGS := -T
$(TOP)/3rdparty/ote_partner/lib/task_linker_script.ld

LOCAL_FORCE_STATIC_EXECUTABLE := true
LOCAL_UNINSTALLABLE_MODULE := true
include $(NVIDIA_EXECUTABLE)

# Make client executable
include $(NVIDIA_DEFAULTS)
LOCAL_MODULE_TAGS := nvidia_tests
LOCAL_C_INCLUDES := $(LOCAL_PATH) \
                    $(TOP)/3rdparty/ote_partner/lib/include \
                    $(TEGRA_TOP)/core/include
LOCAL_SRC_FILES := client_storagedemo.c
LOCAL_STATIC_LIBRARIES := \
    libtlk_common \
    libtlk_client
LOCAL_SHARED_LIBRARIES := libnvos
LOCAL_MODULE := tlkstoragedemo_client

include $(NVIDIA_EXECUTABLE)

endif # SECURE_OS_BUILD == tlk
```

A module like this generates an executable at the following location:

```
$OUT/obj/EXECUTABLES/nvidia_tests_intermediates/tlkstoragedemo_client
```

## Building TLK

TLK builds after the tasks have been compiled. This is because the tasks are post-linked into the kernel to generate a complete image of TLK-plus-tasks. The dependencies in the `Android.mk` makefiles handle this ordering.

The `TASK_MODULES` variable in the `tlk/Android.mk` selects which tasks to post-link into the TLK. This list of modules is used to produce a list of executable files in:

```
$OUT/obj/EXECUTABLES/<task_module>_intermediates/<task_module>
```

The executable files get passed to the TLK make process.

TLK gets built and post-linked with the executable files provided by the modules in `TASK_MODULES` to produce a `tos.img` file. This file is the complete TLK-plus-task image that will be flashed to the TOS partition on the device.

### To build TLK

- In a terminal window, enter:

```
cd tlk
TARGET=<platform> make -e
```

Where `<platform>` is `t132`.

## Adding a Secure Service to a TLK Task

Adding a new task involves modifying the makefile located at:

```
$TOP/3rdparty/ote_partner/tlk/Android.mk
```

### To Add a New Task

- Add the task name to the list of tasks in the `TASK_MODULE` variable of the `Android.mk` file.

The task name is the `LOCAL_MODULE` name for the task you wish to add. For example:

- For the crypto, the name is `tlkcryptodemo_task`
- For the storage sample, the name is `tlkstoragedemo_task`
- The same tasks are not added by default.

- Every time a task is updated; the `tos.img` must be regenerated and flashed onto the device.
- The client binaries, used to trigger an action in the task, are not built by default.
- The Android side binary trigger is performed through the adb shell.

### To add a new task (Example)

1. From a shell prompt execute the following commands:

```
mm
adb remount
adb sync
```

The adb sync pushes the newly compiled binary to the device.

2. Invoke the binary to examine the adb shell.
3. Whenever the client is updated, push it to the device.

### To flash the TLK

- To flash only the `tos.img` file use the following command:

```
$ sudo $(which nvflash) -bl $OUT/bootloader.bin -download TOS
$OUT/tos.img --go
```

**Note:** The following command can only be used to trigger a TLK build:

```
mp tlk
```

## Manifests for Trusted Applications

Each Trusted Application (TA) should have a corresponding manifest that specifies various configuration properties for the TA. The manifest is described by the following data structure:

```
/*
 * Layout of .ote.manifest section in the trusted
 * application is the required UUID followed by an arbitrary
 * number of configuration options.
 */
typedef struct {
    te_service_id_t uuid;
    uint32_t configOptions[];
} OTE_MANIFEST;
```

Each TA must declare and initialize an instance of this `OTE_MANIFEST` structure so that its contents can be placed in a well-defined section in the TA ELF binary where it can then be extracted and processed by TLK during TA initialization.

The following table lists the currently supported configuration options.

Configuration Option	Description
<code>OTE_CONFIG_KEY_MIN_STACK_SIZE</code>	<p>Specifies the desired minimum stack size in bytes for use by TA tasks. The specified value is rounded up to the next page size (e.g., 4 K bytes).</p> <p>Use the <code>OTE_CONFIG_MIN_STACK_SIZE(sz)</code> macro to specify this option. The default for this option is one page (e.g., 4 K bytes).</p>
<code>OTE_CONFIG_KEY_MIN_HEAP_SIZE</code>	<p>Specifies the desired minimum heap size in bytes for use by TA tasks. The specified value is rounded up to the next page size (e.g., 4 K bytes).</p> <p>Use the <code>OTE_CONFIG_MIN_HEAP_SIZE(sz)</code> macro to specify this option. The default value for this option is four pages (e.g., 16 K bytes).</p>
<code>OTE_CONFIG_KEY_MAP_MEM</code>	<p>Enables the mapping of I/O regions owned by the TA to other (secure) TA clients. Use the <code>OTE_CONFIG_MAP_MEM(id,off,sz)</code> macro to enable these mappings.</p> <p>The <code>OTE_CONFIG_MAP_MEM(id,off,sz)</code> macro accepts these arguments:</p> <ul style="list-style-type: none"> <li>• <code>id</code>—specifies a unique 32-bit identifier for the region that is used as a handle when setting up a mapping.</li> <li>• <code>off</code>—specifies the 32-bit physical offset of the region.</li> <li>• <code>sz</code>—specifies the size in bytes of the region. The specified value is rounded up to the next page size (e.g., 4 K bytes).</li> </ul>
<code>OTE_CONFIG_RESTRICT_ACCESS</code> ( <code>&lt;what_to_block&gt;</code> )	<p>Prevents certain types of clients from opening a connection or communicating with the TA. The macro takes an argument that describes which types of applications to block.</p> <p>To block CAs or non-secure applications:  <code>OTE_CONFIG_RESTRICT_ACCESS(OTE_RESTRICT_NON_SECURE_APPS)</code></p> <p>To block TAs but not CAs:  <code>OTE_CONFIG_RESTRICT_ACCESS(OTE_RESTRICT_</code></p>



	SECURE_TASKS)  To restrict all applications (no CAs or TAs): OTE_CONFIG_RESTRICT_ACCESS(OTE_RESTRICT_ SECURE_TASKS   OTE_RESTRICT_NON_SECURE_APPS)
--	---

## Example Manifest

The following shows the contents of an example manifest (`manifest.c`).

```
#include <service/ote_manifest.h>

OTE_MANIFEST OTE_MANIFEST_ATTRS ote_manifest =
{
    /* UUID : {32456890-8572-4a6f-a1f1-03aa9b05f9ff} */
    { 0x32456890, 0x8572, 0x4a6f,
      { 0xa1, 0xf1, 0x03, 0xaa, 0x9b, 0x05, 0xf9, 0xff } },

    /* optional configuration options here */
    {
        /* four pages for heap */
        OTE_CONFIG_MIN_HEAP_SIZE(4 * 4096),
    },
};
```

# Legal Information

## NVIDIA Legal Information

### Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OR CONDITION OF TITLE, MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE AND ON-INFRINGEMENT, ARE HEREBY EXCLUDED TO THE MAXIMUM EXTENT PERMITTED BY LAW.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

### Trademarks

NVIDIA, the NVIDIA logo, and Tegra are trademarks or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

ARM, AMBA, and ARM Powered are registered trademarks of ARM Limited. Cortex, MPCore and Mali are trademarks of ARM Limited. All other brands or product names are the property of their respective holders. "ARM" is used to represent ARM Holdings plc; its operating company ARM Limited; and the regional subsidiaries ARM Inc.; ARM KK; ARM Korea Limited.; ARM Taiwan Limited; ARM France SAS; ARM Consulting (Shanghai) Co. Ltd.; ARM Germany GmbH; ARM Embedded Technologies Pvt. Ltd.; ARM Norway, AS and ARM Sweden AB.

### Copyright

Copyright (c) 2013-2014, NVIDIA CORPORATION. All rights reserved.

## Open Source License

This NVIDIA product contain third party software that is being made available to you under its respective open source software license. That license requires specific legal information to be included in the product.

### Little Kernel License

Little Kernel (LK) is provided under the following terms:

Copyright (c) 2008-2010 Travis Geiselbrecht

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# NVIDIA Tegra BSP for Android TLK FOSS API

Generated by NVIDIA

June 23, 2014

# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>Module Index</b>	<b>3</b>
2.1	Modules . . . . .	3
<b>3</b>	<b>Data Structure Index</b>	<b>5</b>
3.1	Data Structures . . . . .	5
<b>4</b>	<b>File Index</b>	<b>7</b>
4.1	File List . . . . .	7
<b>5</b>	<b>Module Documentation</b>	<b>9</b>
5.1	Trusted Little Kernel (TLK) . . . . .	9
5.1.1	Detailed Description . . . . .	9
5.2	Common Declarations . . . . .	11
5.2.1	Detailed Description . . . . .	11
5.3	Memory/Cache Management . . . . .	12
5.3.1	Detailed Description . . . . .	12
5.3.2	Enumeration Type Documentation . . . . .	12
5.3.2.1	anonymous enum . . . . .	12
5.3.2.2	te_ext_nv_cache_maint_op_t . . . . .	13
5.3.3	Function Documentation . . . . .	13
5.3.3.1	te_ext_nv_cache_maint . . . . .	13
5.3.3.2	te_ext_nv_virt_to_phys . . . . .	13
5.3.3.3	te_ext_nv_get_map_addr . . . . .	14
5.4	Trusted Application (TA) Services . . . . .	15
5.4.1	Detailed Description . . . . .	15
5.5	Client Application Interface . . . . .	17
5.5.1	Detailed Description . . . . .	17
5.5.2	Macro Definition Documentation . . . . .	17
5.5.2.1	TLK_DEVICE_BASE_NAME . . . . .	17
5.5.2.2	TE_IOCTL_MAGIC_NUMBER . . . . .	17
5.5.2.3	TE_IOCTL_OPEN_CLIENT_SESSION . . . . .	17

5.5.2.4	TE_IOCTL_CLOSE_CLIENT_SESSION	17
5.5.2.5	TE_IOCTL_LAUNCH_OP	18
5.5.2.6	TE_IOCTL_SS_NEW_REQ	18
5.5.2.7	TE_IOCTL_SS_REQ_COMPLETE	18
5.5.3	Enumeration Type Documentation	18
5.5.3.1	anonymous enum	18
5.6	User Application Communication	19
5.6.1	Detailed Description	19
5.7	File Handling	20
5.7.1	Detailed Description	20
5.7.2	Macro Definition Documentation	21
5.7.2.1	OTE_MAX_DIR_NAME_LEN	21
5.7.2.2	OTE_MAX_FILE_NAME_LEN	21
5.7.2.3	OTE_MAX_DATA_SIZE	21
5.7.2.4	OTE_RPMB_FRAME_SIZE	21
5.7.2.5	SS_OP_MAX_DATA_SIZE	21
5.7.3	Enumeration Type Documentation	21
5.7.3.1	anonymous enum	21
5.7.3.2	anonymous enum	21
5.7.3.3	anonymous enum	21
5.8	Commands	22
5.8.1	Detailed Description	22
5.8.2	Function Documentation	22
5.8.2.1	te_open_session	22
5.8.2.2	te_close_session	23
5.8.2.3	te_create_operation	23
5.8.2.4	te_init_operation	23
5.8.2.5	te_deinit_operation	23
5.8.2.6	te_launch_operation	23
5.8.2.7	te_oper_set_command	24
5.8.2.8	te_oper_set_param_int_ro	24
5.8.2.9	te_oper_set_param_int_rw	24
5.8.2.10	te_oper_set_param_mem_ro	24
5.8.2.11	te_oper_set_param_mem_rw	24
5.8.2.12	te_oper_get_command	25
5.8.2.13	te_oper_get_num_params	25
5.8.2.14	te_oper_get_param_type	25
5.8.2.15	te_oper_get_param_int	25
5.8.2.16	te_oper_get_param_mem	26
5.8.2.17	te_operation_reset	26

5.9	Types	27
5.9.1	Detailed Description	27
5.9.2	Macro Definition Documentation	27
5.9.2.1	OTE_TASK_NAME_MAX_LENGTH	27
5.9.2.2	OTE_TASK_PRIVATE_DATA_LENGTH	28
5.9.3	Typedef Documentation	28
5.9.3.1	cmnptr_t	28
5.9.4	Enumeration Type Documentation	28
5.9.4.1	te_oper_param_type_t	28
5.9.5	Function Documentation	28
5.9.5.1	te_get_result_origin	28
5.10	Errors	29
5.10.1	Detailed Description	29
5.10.2	Enumeration Type Documentation	29
5.10.2.1	te_error_t	29
5.10.2.2	te_result_origin_t	30
5.11	Common TA Service Attributes	31
5.11.1	Detailed Description	31
5.11.2	Macro Definition Documentation	32
5.11.2.1	OTE_ATTR_VAL	32
5.11.2.2	OTE_ATTR_PUB	32
5.11.3	Enumeration Type Documentation	32
5.11.3.1	te_attribute_id_t	32
5.11.4	Function Documentation	33
5.11.4.1	te_set_mem_attribute	33
5.11.4.2	te_get_mem_attribute_buffer	33
5.11.4.3	te_get_mem_attribute_size	33
5.11.4.4	te_copy_mem_attribute	33
5.11.4.5	te_set_int_attribute	34
5.11.4.6	te_free_internal_attribute	34
5.11.4.7	te_copy_attribute	34
5.12	Crypto Service	35
5.12.1	Detailed Description	35
5.12.2	Typedef Documentation	36
5.12.2.1	te_crypto_object_t	36
5.12.2.2	te_crypto_operation_t	36
5.12.3	Enumeration Type Documentation	36
5.12.3.1	te_oper_crypto_algo_t	36
5.12.3.2	te_oper_crypto_algo_mode_t	37
5.12.4	Function Documentation	37

5.12.4.1	<a href="#">te_allocate_object</a>	37
5.12.4.2	<a href="#">te_populate_object</a>	37
5.12.4.3	<a href="#">te_free_object</a>	37
5.12.4.4	<a href="#">te_allocate_operation</a>	38
5.12.4.5	<a href="#">te_set_operation_key</a>	38
5.12.4.6	<a href="#">te_cipher_init</a>	38
5.12.4.7	<a href="#">te_cipher_update</a>	38
5.12.4.8	<a href="#">te_cipher_do_final</a>	39
5.12.4.9	<a href="#">te_rsa_init</a>	39
5.12.4.10	<a href="#">te_rsa_handle_request</a>	39
5.12.4.11	<a href="#">te_free_operation</a>	39
5.12.4.12	<a href="#">te_generate_random</a>	40
5.12.4.13	<a href="#">te_get_attribute_by_id</a>	40
5.13	<a href="#">Manifest Layout</a>	41
5.13.1	<a href="#">Detailed Description</a>	41
5.13.2	<a href="#">Macro Definition Documentation</a>	41
5.13.2.1	<a href="#">OTE_CONFIG_MIN_STACK_SIZE</a>	41
5.13.2.2	<a href="#">OTE_CONFIG_MIN_HEAP_SIZE</a>	42
5.13.2.3	<a href="#">OTE_CONFIG_MAP_MEM</a>	42
5.13.2.4	<a href="#">OTE_CONFIG_RESTRICT_ACCESS</a>	42
5.13.2.5	<a href="#">OTE_CONFIG_AUTHORIZE</a>	42
5.13.2.6	<a href="#">OTE_CONFIG_TASK_INITIAL_STATE</a>	42
5.13.2.7	<a href="#">OTE_MANIFEST_ATTRS</a>	43
5.13.3	<a href="#">Enumeration Type Documentation</a>	43
5.13.3.1	<a href="#">ote_config_key_t</a>	43
5.13.3.2	<a href="#">anonymous enum</a>	43
5.13.3.3	<a href="#">anonymous enum</a>	43
5.13.3.4	<a href="#">anonymous enum</a>	43
5.14	<a href="#">Memory Allocation</a>	44
5.14.1	<a href="#">Detailed Description</a>	44
5.14.2	<a href="#">Function Documentation</a>	44
5.14.2.1	<a href="#">te_mem_alloc</a>	44
5.14.2.2	<a href="#">te_mem_calloc</a>	44
5.14.2.3	<a href="#">te_mem_free</a>	45
5.14.2.4	<a href="#">te_mem_fill</a>	45
5.14.2.5	<a href="#">te_mem_move</a>	45
5.14.2.6	<a href="#">te_mem_compare</a>	45
5.15	<a href="#">Crypto Service Manager</a>	46
5.15.1	<a href="#">Detailed Description</a>	46
5.15.2	<a href="#">Function Documentation</a>	46



5.15.2.1	<a href="#">ote_nvcrypto_init</a>	46
5.15.2.2	<a href="#">ote_nvcrypto_deinit</a>	46
5.15.2.3	<a href="#">ote_nvcrypto_get_keybox</a>	46
5.15.2.4	<a href="#">ote_nvcrypto_get_storage_key</a>	47
5.16	OTF Decoder Service	48
5.16.1	Detailed Description	48
5.16.2	Function Documentation	48
5.16.2.1	<a href="#">ote_otf_init</a>	48
5.16.2.2	<a href="#">ote_otf_deinit</a>	48
5.16.2.3	<a href="#">ote_otf_setkey</a>	49
5.16.2.4	<a href="#">ote_otf_erasekey</a>	49
5.17	RTC Service	50
5.17.1	Detailed Description	50
5.17.2	Function Documentation	50
5.17.2.1	<a href="#">ote_rtc_init</a>	50
5.17.2.2	<a href="#">ote_rtc_deinit</a>	50
5.17.2.3	<a href="#">ote_rtc_get_time</a>	50
5.18	Interface	52
5.18.1	Detailed Description	52
5.18.2	Macro Definition Documentation	53
5.18.2.1	<a href="#">DEVICE_UID_SIZE_BYTES</a>	53
5.18.2.2	<a href="#">OTE_TE_FPRINTF_PREFIX_MAX_LENGTH</a>	53
5.18.3	Enumeration Type Documentation	53
5.18.3.1	<a href="#">anonymous enum</a>	53
5.18.3.2	<a href="#">anonymous enum</a>	54
5.18.3.3	<a href="#">anonymous enum</a>	54
5.18.3.4	<a href="#">anonymous enum</a>	54
5.18.3.5	<a href="#">te_property_type_t</a>	54
5.18.4	Function Documentation	54
5.18.4.1	<a href="#">te_exit_service</a>	54
5.18.4.2	<a href="#">te_init</a>	54
5.18.4.3	<a href="#">te_destroy</a>	55
5.18.4.4	<a href="#">te_create_instance_iface</a>	55
5.18.4.5	<a href="#">te_destroy_instance_iface</a>	55
5.18.4.6	<a href="#">te_open_session_iface</a>	55
5.18.4.7	<a href="#">te_close_session_iface</a>	55
5.18.4.8	<a href="#">te_receive_operation_iface</a>	55
5.18.4.9	<a href="#">ote_get_instance_data</a>	55
5.18.4.10	<a href="#">ote_set_instance_data</a>	55
5.18.4.11	<a href="#">te_get_current_ta_uuid</a>	55

5.18.4.12	<a href="#">te_get_client_ta_identity</a>	56
5.18.4.13	<a href="#">te_get_device_unique_id</a>	56
5.18.4.14	<a href="#">te_fprintf_set_prefix</a>	56
5.18.4.15	<a href="#">te_fprintf</a>	56
5.18.4.16	<a href="#">te_oper_dump_param</a>	56
5.18.4.17	<a href="#">te_oper_dump_param_list</a>	57
5.19	<a href="#">Storage Service</a>	58
5.19.1	<a href="#">Detailed Description</a>	58
5.19.2	<a href="#">Macro Definition Documentation</a>	59
5.19.2.1	<a href="#">TE_STORAGE_OBJID_MAX_LEN</a>	59
5.19.2.2	<a href="#">TE_MAX_STORAGE_REQUEST_PARAMS</a>	59
5.19.3	<a href="#">Typedef Documentation</a>	59
5.19.3.1	<a href="#">te_storage_object_t</a>	59
5.19.4	<a href="#">Enumeration Type Documentation</a>	59
5.19.4.1	<a href="#">anonymous enum</a>	59
5.19.4.2	<a href="#">te_storage_flags_t</a>	60
5.19.4.3	<a href="#">te_storage_whence_t</a>	60
5.19.5	<a href="#">Function Documentation</a>	60
5.19.5.1	<a href="#">te_create_storage_object</a>	60
5.19.5.2	<a href="#">te_open_storage_object</a>	60
5.19.5.3	<a href="#">te_read_storage_object</a>	61
5.19.5.4	<a href="#">te_write_storage_object</a>	61
5.19.5.5	<a href="#">te_get_storage_object_size</a>	61
5.19.5.6	<a href="#">te_seek_storage_object</a>	61
5.19.5.7	<a href="#">te_trunc_storage_object</a>	62
5.19.5.8	<a href="#">te_delete_storage_object</a>	62
5.19.5.9	<a href="#">te_close_storage_object</a>	62
5.20	<a href="#">Task Loader</a>	63
5.20.1	<a href="#">Detailed Description</a>	63
5.20.2	<a href="#">Typedef Documentation</a>	64
5.20.2.1	<a href="#">te_list_apps_t</a>	64
5.20.2.2	<a href="#">te_task_request_args_t</a>	64
5.20.3	<a href="#">Enumeration Type Documentation</a>	64
5.20.3.1	<a href="#">te_get_info_type_t</a>	64
5.20.3.2	<a href="#">te_app_id_t</a>	65
5.20.3.3	<a href="#">te_task_opcode_t</a>	65
5.20.4	<a href="#">Function Documentation</a>	65
5.20.4.1	<a href="#">te_app_request_memory</a>	65
5.20.4.2	<a href="#">te_app_prepare</a>	65
5.20.4.3	<a href="#">te_app_start</a>	66

5.20.4.4	te_list_apps	66
5.20.4.5	te_task_get_info	66
5.20.4.6	te_task_get_mapping	67
5.20.4.7	te_get_pending_task_mapping	67
5.20.4.8	te_task_get_name_self	68
5.20.4.9	te_task_system_info	68
5.20.4.10	te_app_unload	68
5.20.4.11	te_app_block	68
5.20.4.12	te_app_unblock	68
<b>6</b>	<b>Data Structure Documentation</b>	<b>71</b>
6.1	__te_crypto_operation_t Struct Reference	71
6.1.1	Detailed Description	71
6.1.2	Field Documentation	72
6.1.2.1	info	72
6.1.2.2	key	72
6.1.2.3	iv	72
6.1.2.4	iv_len	72
6.1.2.5	imp_obj	72
6.1.2.6	init	72
6.1.2.7	update	72
6.1.2.8	do_final	72
6.1.2.9	handle_req	72
6.1.2.10	free	72
6.2	ote_closesession Struct Reference	72
6.2.1	Detailed Description	72
6.2.2	Field Documentation	72
6.2.2.1	session_id	72
6.2.2.2	answer	72
6.3	ote_file_close_params_t Struct Reference	72
6.3.1	Field Documentation	73
6.3.1.1	handle	73
6.4	ote_file_create_params_t Struct Reference	73
6.4.1	Field Documentation	73
6.4.1.1	dname	73
6.4.1.2	fname	73
6.4.1.3	flags	73
6.5	ote_file_delete_params_t Struct Reference	73
6.5.1	Field Documentation	73
6.5.1.1	dname	73

6.5.1.2	fname	73
6.6	ote_file_get_size_params_t Struct Reference	74
6.6.1	Field Documentation	74
6.6.1.1	handle	74
6.6.1.2	size	74
6.7	ote_file_open_params_t Struct Reference	74
6.7.1	Field Documentation	74
6.7.1.1	dname	74
6.7.1.2	fname	74
6.7.1.3	flags	74
6.7.1.4	handle	74
6.8	ote_file_read_params_t Struct Reference	74
6.8.1	Field Documentation	75
6.8.1.1	handle	75
6.8.1.2	data_size	75
6.8.1.3	data	75
6.9	ote_file_req_params_t Union Reference	75
6.9.1	Field Documentation	76
6.9.1.1	f_create	76
6.9.1.2	f_delete	76
6.9.1.3	f_open	76
6.9.1.4	f_close	76
6.9.1.5	f_read	76
6.9.1.6	f_write	76
6.9.1.7	f_seek	76
6.9.1.8	f_trunc	76
6.9.1.9	f_getsize	76
6.9.1.10	f_rpmb_write	76
6.9.1.11	f_rpmb_read	76
6.10	ote_file_req_t Struct Reference	77
6.10.1	Field Documentation	77
6.10.1.1	type	77
6.10.1.2	result	77
6.10.1.3	params_size	77
6.10.1.4	params	77
6.11	ote_file_rpmb_read_params_t Struct Reference	77
6.11.1	Field Documentation	78
6.11.1.1	req_frame	78
6.11.1.2	resp_frame	78
6.12	ote_file_rpmb_write_params_t Struct Reference	78

6.12.1	Field Documentation	78
6.12.1.1	req_frame	78
6.12.1.2	req_resp_frame	78
6.12.1.3	resp_frame	78
6.13	ote_file_seek_params_t Struct Reference	78
6.13.1	Field Documentation	78
6.13.1.1	handle	78
6.13.1.2	offset	78
6.13.1.3	whence	78
6.14	ote_file_trunc_params_t Struct Reference	79
6.14.1	Field Documentation	79
6.14.1.1	handle	79
6.14.1.2	length	79
6.15	ote_file_write_params_t Struct Reference	79
6.15.1	Field Documentation	79
6.15.1.1	handle	79
6.15.1.2	data_size	79
6.15.1.3	data	79
6.16	ote_launchop Struct Reference	79
6.16.1	Detailed Description	79
6.16.2	Field Documentation	80
6.16.2.1	session_id	80
6.16.2.2	operation	80
6.16.2.3	answer	80
6.17	OTE_MANIFEST Struct Reference	80
6.17.1	Detailed Description	80
6.17.2	Field Documentation	81
6.17.2.1	name	81
6.17.2.2	private_data	81
6.17.2.3	uuid	81
6.17.2.4	config_options	81
6.18	ote_opensession Struct Reference	81
6.18.1	Detailed Description	81
6.18.2	Field Documentation	82
6.18.2.1	dest_service_id	82
6.18.2.2	operation	82
6.18.2.3	answer	82
6.19	ote_ss_op_t Struct Reference	82
6.19.1	Field Documentation	82
6.19.1.1	req_size	82

6.19.1.2	data	82
6.20	te_answer Struct Reference	83
6.20.1	Field Documentation	83
6.20.1.1	result	83
6.20.1.2	session_id	83
6.20.1.3	result_origin	83
6.21	te_app_block_args_t Struct Reference	83
6.21.1	Field Documentation	83
6.21.1.1	tid_type	83
6.21.1.2	tid_index	84
6.21.1.3	tid_uuid	84
6.21.1.4	"@33	84
6.22	te_app_block_t Struct Reference	84
6.22.1	Field Documentation	84
6.22.1.1	aid_type	84
6.22.1.2	aid_index	84
6.22.1.3	aid_uuid	84
6.22.1.4	"@35	84
6.23	te_app_list_args_t Struct Reference	85
6.23.1	Detailed Description	85
6.23.2	Field Documentation	85
6.23.2.1	app_index	85
6.23.2.2	app_type	85
6.23.2.3	app_state	85
6.23.2.4	app_info	85
6.24	te_app_load_memory_request_args_t Struct Reference	85
6.24.1	Detailed Description	86
6.24.2	Field Documentation	86
6.24.2.1	app_handle	86
6.24.2.2	app_addr	86
6.24.2.3	app_size	86
6.25	te_app_prepare_args_t Struct Reference	86
6.25.1	Detailed Description	86
6.25.2	Field Documentation	87
6.25.2.1	app_handle	87
6.25.2.2	te_task_info	87
6.26	te_app_start_args_t Struct Reference	87
6.26.1	Detailed Description	87
6.26.2	Field Documentation	87
6.26.2.1	app_handle	87

6.26.2.2	<a href="#">app_reject</a>	87
6.26.2.3	<a href="#">app_restrictions</a>	87
6.27	<a href="#">te_app_unload_args_t Struct Reference</a>	88
6.27.1	<a href="#">Field Documentation</a>	88
6.27.1.1	<a href="#">tid_type</a>	88
6.27.1.2	<a href="#">tid_index</a>	88
6.27.1.3	<a href="#">tid_uuid</a>	88
6.27.1.4	<a href="#">"@29</a>	88
6.28	<a href="#">te_app_unload_t Struct Reference</a>	89
6.28.1	<a href="#">Field Documentation</a>	89
6.28.1.1	<a href="#">aid_type</a>	89
6.28.1.2	<a href="#">aid_index</a>	89
6.28.1.3	<a href="#">aid_uuid</a>	89
6.28.1.4	<a href="#">"@31</a>	89
6.29	<a href="#">te_attribute_t Struct Reference</a>	89
6.29.1	<a href="#">Detailed Description</a>	89
6.29.2	<a href="#">Field Documentation</a>	90
6.29.2.1	<a href="#">id</a>	90
6.29.2.2	<a href="#">buffer</a>	90
6.29.2.3	<a href="#">size</a>	90
6.29.2.4	<a href="#">ref</a>	90
6.29.2.5	<a href="#">a</a>	90
6.29.2.6	<a href="#">b</a>	90
6.29.2.7	<a href="#">value</a>	90
6.29.2.8	<a href="#">content</a>	90
6.30	<a href="#">te_cache_maint_args_t Struct Reference</a>	90
6.30.1	<a href="#">Detailed Description</a>	90
6.30.2	<a href="#">Field Documentation</a>	91
6.30.2.1	<a href="#">vaddr</a>	91
6.30.2.2	<a href="#">length</a>	91
6.30.2.3	<a href="#">op</a>	91
6.31	<a href="#">te_cmd Union Reference</a>	91
6.31.1	<a href="#">Field Documentation</a>	91
6.31.1.1	<a href="#">opensession</a>	91
6.31.1.2	<a href="#">closesession</a>	91
6.31.1.3	<a href="#">launchop</a>	92
6.32	<a href="#">te_crypto_operation_info_t Struct Reference</a>	92
6.32.1	<a href="#">Detailed Description</a>	92
6.32.2	<a href="#">Field Documentation</a>	92
6.32.2.1	<a href="#">algorithm</a>	92

6.32.2.2	<a href="#">operation_class</a>	92
6.32.2.3	<a href="#">mode</a>	92
6.32.2.4	<a href="#">digest_length</a>	92
6.32.2.5	<a href="#">key_size</a>	92
6.32.2.6	<a href="#">required_key_usage</a>	92
6.32.2.7	<a href="#">handle_state</a>	92
6.33	<a href="#">te_crypto_rsa_key_t Struct Reference</a>	92
6.33.1	<a href="#">Detailed Description</a>	92
6.33.2	<a href="#">Field Documentation</a>	93
6.33.2.1	<a href="#">public_mod</a>	93
6.33.2.2	<a href="#">public_mod_len</a>	93
6.33.2.3	<a href="#">public_expo</a>	93
6.33.2.4	<a href="#">public_expo_len</a>	93
6.33.2.5	<a href="#">private_expo</a>	93
6.33.2.6	<a href="#">private_expo_len</a>	93
6.33.2.7	<a href="#">prime1</a>	93
6.33.2.8	<a href="#">prime1_len</a>	93
6.33.2.9	<a href="#">prime2</a>	94
6.33.2.10	<a href="#">prime2_len</a>	94
6.33.2.11	<a href="#">expo1</a>	94
6.33.2.12	<a href="#">expo1_len</a>	94
6.33.2.13	<a href="#">expo2</a>	94
6.33.2.14	<a href="#">expo2_len</a>	94
6.33.2.15	<a href="#">coeff</a>	94
6.33.2.16	<a href="#">coeff_len</a>	94
6.34	<a href="#">te_device_unique_id Struct Reference</a>	94
6.34.1	<a href="#">Detailed Description</a>	94
6.34.2	<a href="#">Field Documentation</a>	95
6.34.2.1	<a href="#">id</a>	95
6.35	<a href="#">te_entry_point_message_t Struct Reference</a>	95
6.35.1	<a href="#">Field Documentation</a>	95
6.35.1.1	<a href="#">type</a>	95
6.35.1.2	<a href="#">result</a>	95
6.35.1.3	<a href="#">context</a>	95
6.35.1.4	<a href="#">command_id</a>	95
6.35.1.5	<a href="#">params</a>	95
6.35.1.6	<a href="#">params_size</a>	95
6.36	<a href="#">te_get_pending_map_args_t Struct Reference</a>	96
6.36.1	<a href="#">Field Documentation</a>	96
6.36.1.1	<a href="#">pm_handle</a>	96



6.36.1.2	pm_map	96
6.37	te_get_property_args_t Struct Reference	96
6.37.1	Detailed Description	96
6.37.2	Field Documentation	97
6.37.2.1	prop	97
6.37.2.2	data_type	97
6.37.2.3	uuid	97
6.37.2.4	identity	97
6.37.2.5	value	98
6.37.2.6	value_size	98
6.37.2.7	result	98
6.38	te_get_task_info_t Struct Reference	98
6.38.1	Field Documentation	99
6.38.1.1	gti_request_type	99
6.38.1.2	gtiu_index	99
6.38.1.3	gtiu_uuid	99
6.38.1.4	"@25	99
6.38.1.5	gti_state	99
6.38.1.6	gti_type	99
6.38.1.7	gti_info	99
6.39	te_get_task_mapping_t Struct Reference	99
6.39.1	Field Documentation	99
6.39.1.1	gmt_request_type	99
6.39.1.2	gmtu_index	100
6.39.1.3	gmtu_uuid	100
6.39.1.4	"@27	100
6.39.1.5	gmt_map	100
6.40	te_identity_t Struct Reference	100
6.40.1	Detailed Description	100
6.40.2	Field Documentation	100
6.40.2.1	login	100
6.40.2.2	uuid	100
6.41	te_list_apps Struct Reference	100
6.41.1	Detailed Description	101
6.41.2	Field Documentation	101
6.41.2.1	al_index	101
6.41.2.2	al_type	101
6.41.2.3	al_state	101
6.41.2.4	al_info	101
6.42	te_map_mem_addr_args_t Struct Reference	101

6.42.1 Detailed Description . . . . .	101
6.42.2 Field Documentation . . . . .	102
6.42.2.1 id . . . . .	102
6.42.2.2 addr . . . . .	102
6.43 te_memory_mapping_t Struct Reference . . . . .	102
6.43.1 Field Documentation . . . . .	102
6.43.1.1 map_index . . . . .	102
6.43.1.2 map_id . . . . .	102
6.43.1.3 map_offset . . . . .	102
6.43.1.4 map_size . . . . .	102
6.44 te_oper_param_t Struct Reference . . . . .	102
6.44.1 Detailed Description . . . . .	102
6.44.2 Field Documentation . . . . .	103
6.44.2.1 index . . . . .	103
6.44.2.2 type . . . . .	103
6.44.2.3 val . . . . .	103
6.44.2.4 Int . . . . .	103
6.44.2.5 base . . . . .	103
6.44.2.6 len . . . . .	103
6.44.2.7 Mem . . . . .	103
6.44.2.8 u . . . . .	103
6.44.2.9 next . . . . .	103
6.45 te_operation_t Struct Reference . . . . .	103
6.45.1 Detailed Description . . . . .	103
6.45.2 Field Documentation . . . . .	104
6.45.2.1 command . . . . .	104
6.45.2.2 status . . . . .	104
6.45.2.3 list_head . . . . .	104
6.45.2.4 list_tail . . . . .	104
6.45.2.5 list_count . . . . .	104
6.45.2.6 interface_side . . . . .	104
6.46 te_request_t Struct Reference . . . . .	104
6.46.1 Detailed Description . . . . .	104
6.46.2 Field Documentation . . . . .	104
6.46.2.1 type . . . . .	104
6.46.2.2 session_id . . . . .	104
6.46.2.3 command_id . . . . .	104
6.46.2.4 params . . . . .	104
6.46.2.5 params_size . . . . .	105
6.46.2.6 dest_uuid . . . . .	105

6.46.2.7	result	105
6.46.2.8	result_origin	105
6.47	te_service_id_t Struct Reference	105
6.47.1	Detailed Description	105
6.47.2	Field Documentation	105
6.47.2.1	time_low	105
6.47.2.2	time_mid	105
6.47.2.3	time_hi_and_version	105
6.47.2.4	clock_seq_and_node	105
6.48	te_session_t Union Reference	105
6.48.1	Detailed Description	105
6.48.2	Field Documentation	106
6.48.2.1	session_id	106
6.48.2.2	context_id	106
6.48.2.3	result_origin	106
6.48.2.4	client	106
6.48.2.5	service	106
6.49	te_storage_param_t Union Reference	106
6.49.1	Detailed Description	106
6.49.2	Field Documentation	106
6.49.2.1	a	106
6.49.2.2	val	106
6.49.2.3	base	106
6.49.2.4	len	106
6.49.2.5	mem	107
6.50	te_storage_request_t Struct Reference	107
6.50.1	Detailed Description	107
6.50.2	Field Documentation	107
6.50.2.1	type	107
6.50.2.2	args	107
6.50.2.3	result	107
6.51	te_system_info_args_t Struct Reference	108
6.51.1	Field Documentation	108
6.51.1.1	si_type	108
6.52	te_ta_to_ta_request_args_t Struct Reference	108
6.52.1	Field Documentation	108
6.52.1.1	request	108
6.53	te_task_info_t Struct Reference	108
6.53.1	Detailed Description	108
6.53.2	Field Documentation	109

6.53.2.1	uuid	109
6.53.2.2	manifest_exists	109
6.53.2.3	multi_instance	109
6.53.2.4	min_stack_size	109
6.53.2.5	min_heap_size	109
6.53.2.6	map_io_mem_cnt	109
6.53.2.7	restrict_access	109
6.53.2.8	authorizations	109
6.53.2.9	initial_state	109
6.53.2.10	task_name	109
6.53.2.11	task_private_data	110
6.54	te_task_request_args_s Struct Reference	110
6.54.1	Detailed Description	110
6.54.2	Field Documentation	110
6.54.2.1	ia_opcode	110
6.54.2.2	ia_load_memory_request	110
6.54.2.3	ia_prepare	111
6.54.2.4	ia_pending_mapping	111
6.54.2.5	ia_start	111
6.54.2.6	ia_list	111
6.54.2.7	ia_get_task_info	111
6.54.2.8	ia_get_task_mapping	111
6.54.2.9	ia_app_unload	111
6.54.2.10	ia_app_block	111
6.54.2.11	ia_system_info	111
6.54.2.12	"@37	111
6.55	te_task_restrictions_t Struct Reference	111
6.55.1	Detailed Description	111
6.55.2	Field Documentation	111
6.55.2.1	min_stack_size	111
6.55.2.2	min_heap_size	111
6.55.2.3	task_name	112
6.56	te_v_to_p_args_t Struct Reference	112
6.56.1	Detailed Description	112
6.56.2	Field Documentation	112
6.56.2.1	vaddr	112
6.56.2.2	paddr	112
<b>7</b>	<b>File Documentation</b>	<b>113</b>
7.1	nvtlkoss.txt File Reference	113

7.2	ote_attrs.h File Reference . . . . .	113
7.2.1	Detailed Description . . . . .	113
7.3	ote_client.h File Reference . . . . .	114
7.3.1	Detailed Description . . . . .	114
7.4	ote_command.h File Reference . . . . .	116
7.4.1	Detailed Description . . . . .	116
7.5	ote_common.h File Reference . . . . .	116
7.5.1	Detailed Description . . . . .	116
7.6	ote_crypto.h File Reference . . . . .	117
7.6.1	Detailed Description . . . . .	117
7.7	ote_error.h File Reference . . . . .	118
7.7.1	Detailed Description . . . . .	118
7.8	ote_ext_nv.h File Reference . . . . .	119
7.8.1	Detailed Description . . . . .	119
7.9	ote_manifest.h File Reference . . . . .	120
7.9.1	Detailed Description . . . . .	120
7.10	ote_memory.h File Reference . . . . .	121
7.10.1	Detailed Description . . . . .	121
7.11	ote_nvcrypto.h File Reference . . . . .	121
7.11.1	Detailed Description . . . . .	121
7.12	ote_otf.h File Reference . . . . .	121
7.12.1	Detailed Description . . . . .	121
7.13	ote_rtc.h File Reference . . . . .	122
7.13.1	Detailed Description . . . . .	122
7.14	ote_service.h File Reference . . . . .	122
7.14.1	Detailed Description . . . . .	122
7.15	ote_storage.h File Reference . . . . .	123
7.15.1	Detailed Description . . . . .	123
7.16	ote_task_load.h File Reference . . . . .	124
7.16.1	Detailed Description . . . . .	124



# Chapter 1

## Main Page

Welcome to *NVIDIA® Tegra® BSP for Android TLK Open Source API*. Documentation is preliminary and subject to change.

### API Documentation

The API content of this document was generated from code comments. See the **Module Index** in the navigation pane for a list of the modules included in this documentation.

### Warning

This API content represents the set of APIs you can **use directly**. Some APIs are not included in this document and we recommend you do not use them directly. Using undocumented APIs can lead to incompatibility when upgrading to later releases.





## Chapter 2

# Module Index

### 2.1 Modules

Here is a list of all modules:

Trusted Little Kernel (TLK) . . . . .	9
Client Application Interface . . . . .	17
File Handling . . . . .	20
User Application Communication . . . . .	19
Common Declarations . . . . .	11
Commands . . . . .	22
Errors . . . . .	29
Types . . . . .	27
Memory/Cache Management . . . . .	12
Trusted Application (TA) Services . . . . .	15
Common TA Service Attributes . . . . .	31
Crypto Service . . . . .	35
Crypto Service Manager . . . . .	46
Interface . . . . .	52
Manifest Layout . . . . .	41
Memory Allocation . . . . .	44
OTF Decoder Service . . . . .	48
RTC Service . . . . .	50
Storage Service . . . . .	58
Task Loader . . . . .	63



## Chapter 3

# Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">__te_crypto_operation_t</a>	71
<a href="#">ote_closesession</a>	
Closes an OTE session	72
<a href="#">ote_file_close_params_t</a>	72
<a href="#">ote_file_create_params_t</a>	73
<a href="#">ote_file_delete_params_t</a>	73
<a href="#">ote_file_get_size_params_t</a>	74
<a href="#">ote_file_open_params_t</a>	74
<a href="#">ote_file_read_params_t</a>	74
<a href="#">ote_file_req_params_t</a>	75
<a href="#">ote_file_req_t</a>	77
<a href="#">ote_file_rpmb_read_params_t</a>	77
<a href="#">ote_file_rpmb_write_params_t</a>	78
<a href="#">ote_file_seek_params_t</a>	78
<a href="#">ote_file_trunc_params_t</a>	79
<a href="#">ote_file_write_params_t</a>	79
<a href="#">ote_launchop</a>	
Launches an operation request	79
<a href="#">OTE_MANIFEST</a>	80
<a href="#">ote_opensession</a>	
Opens an open trusted environment (OTE) session	81
<a href="#">ote_ss_op_t</a>	82
<a href="#">te_answer</a>	83
<a href="#">te_app_block_args_t</a>	83
<a href="#">te_app_block_t</a>	84
<a href="#">te_app_list_args_t</a>	
Holds arguments for the ioctl handler	85
<a href="#">te_app_load_memory_request_args_t</a>	85
<a href="#">te_app_prepare_args_t</a>	86
<a href="#">te_app_start_args_t</a>	87
<a href="#">te_app_unload_args_t</a>	88
<a href="#">te_app_unload_t</a>	89
<a href="#">te_attribute_t</a>	89
<a href="#">te_cache_maint_args_t</a>	
Holds an op code and data used to for cache maintenance	90
<a href="#">te_cmd</a>	91
<a href="#">te_crypto_operation_info_t</a>	92
<a href="#">te_crypto_rsa_key_t</a>	92

<a href="#">te_device_unique_id</a>	94
<a href="#">te_entry_point_message_t</a>	95
<a href="#">te_get_pending_map_args_t</a>	96
<a href="#">te_get_property_args_t</a>	96
<a href="#">te_get_task_info_t</a>	98
<a href="#">te_get_task_mapping_t</a>	99
<a href="#">te_identity_t</a>	100
<a href="#">te_list_apps</a>	100
<a href="#">te_map_mem_addr_args_t</a>	
Holds a pointer to the map memory for a specific <a href="#">OTE_CONFIG_MAP_MEM</a> ID value	101
<a href="#">te_memory_mapping_t</a>	102
<a href="#">te_oper_param_t</a>	102
<a href="#">te_operation_t</a>	103
<a href="#">te_request_t</a>	
Holds the layout of the <a href="#">te_oper_param_t</a> structures which must match the layout sent in by the non-secure (NS) world via the TrustZone Secure Monitor Call (TZ SMC) path	104
<a href="#">te_service_id_t</a>	105
<a href="#">te_session_t</a>	105
<a href="#">te_storage_param_t</a>	106
<a href="#">te_storage_request_t</a>	107
<a href="#">te_system_info_args_t</a>	108
<a href="#">te_ta_to_ta_request_args_t</a>	108
<a href="#">te_task_info_t</a>	108
<a href="#">te_task_request_args_s</a>	110
<a href="#">te_task_restrictions_t</a>	111
<a href="#">te_v_to_p_args_t</a>	
Holds a pointer to the physical address for a specific virtual address	112

## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

<a href="#">ote_attrs.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Service Attributes . . . . .</b>	<b>113</b>
<a href="#">ote_client.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Client Communications . . . . .</b>	<b>114</b>
<a href="#">ote_command.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Common Commands . . . . .</b>	<b>116</b>
<a href="#">ote_common.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Common Declarations . . . . .</b>	<b>116</b>
<a href="#">ote_crypto.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Cryptography . . . . .</b>	<b>117</b>
<a href="#">ote_error.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Error Handling . . . . .</b>	<b>118</b>
<a href="#">ote_ext_nv.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Memory/Cache Management . . . . .</b>	<b>119</b>
<a href="#">ote_manifest.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Manifest Layout . . . . .</b>	<b>120</b>
<a href="#">ote_memory.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Memory Functions . . . . .</b>	<b>121</b>
<a href="#">ote_nvcrypto.h</a>	<b>NVIDIA Trusted Little Kernel Interface: NVIDIA Cryptography . . . . .</b>	<b>121</b>
<a href="#">ote_otf.h</a>	<b>NVIDIA Trusted Little Kernel Interface: On-the-Fly (OTF) Decoder Service . . . . .</b>	<b>121</b>
<a href="#">ote_rtc.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Real-Time Clock (RTC) Services . . . . .</b>	<b>122</b>
<a href="#">ote_service.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Service Interface . . . . .</b>	<b>122</b>
<a href="#">ote_storage.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Storage Services . . . . .</b>	<b>123</b>
<a href="#">ote_task_load.h</a>	<b>NVIDIA Trusted Little Kernel Interface: Task-Loading Interface . . . . .</b>	<b>124</b>



## Chapter 5

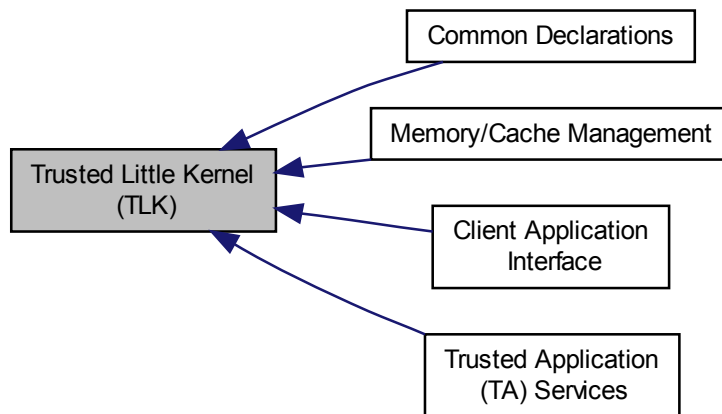
# Module Documentation

### 5.1 Trusted Little Kernel (TLK)

#### 5.1.1 Detailed Description

The NVIDIA® Trusted Little Kernel (TLK) technology extends technology made available with the development of the Little Kernel (LK). A TLK service is a Trusted Application (TA) that provides a service to other TAs. Examples of TLK services are: Crypto and OTF.

The Android OS and Trusted Little Kernel (TLK) software operate in a master-slave relationship, with TLK as the slave. TLK is compiled using the GCC when building your NVIDIA® Tegra® BSP for Android release. TLK resides in a separate partition, which facilitates debugging. Collaboration diagram for Trusted Little Kernel (TLK):



#### Modules

- [Client Application Interface](#)  
*Defines the client application APIs.*
- [Common Declarations](#)  
*Defines the common declarations, functions, and error codes for the TLK.*
- [Memory/Cache Management](#)

*Declarations and functions for TLK memory/cache management.*

- [Trusted Application \(TA\) Services](#)

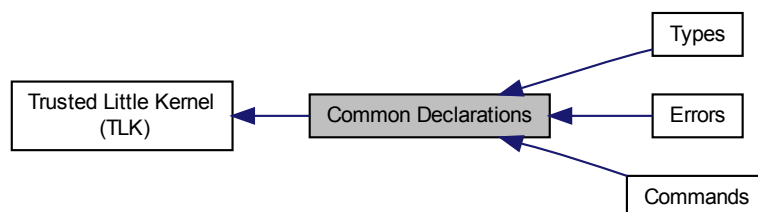
*Declarations and functions for the TA services.*



## 5.2 Common Declarations

### 5.2.1 Detailed Description

Defines the common declarations, functions, and error codes for the TLK. Collaboration diagram for Common Declarations:



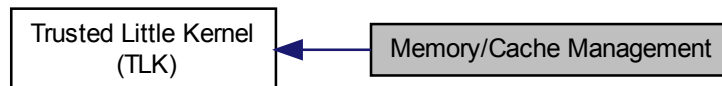
### Modules

- [Commands](#)  
*Defines common functions for Trusted Little Kernel (TLK).*
- [Errors](#)  
*Defines common error codes for Trusted Little Kernel (TLK).*
- [Types](#)  
*Defines common data types and functions for Trusted Little Kernel (TLK).*

## 5.3 Memory/Cache Management

### 5.3.1 Detailed Description

Declarations and functions for TLK memory/cache management. Defines TLK memory and cache management data types and functions. Collaboration diagram for Memory/Cache Management:



### Data Structures

- struct [te\\_map\\_mem\\_addr\\_args\\_t](#)  
*Holds a pointer to the map memory for a specific [OTE\\_CONFIG\\_MAP\\_MEM](#) ID value.*
- struct [te\\_v\\_to\\_p\\_args\\_t](#)  
*Holds a pointer to the physical address for a specific virtual address.*
- struct [te\\_cache\\_maint\\_args\\_t](#)  
*Holds an op code and data used to for cache maintenance.*

### Enumerations

- enum {  
  [OTE\\_IOCTL\\_GET\\_MAP\\_MEM\\_ADDR](#) = 1,  
  [OTE\\_IOCTL\\_V\\_TO\\_P](#) = 2,  
  [OTE\\_IOCTL\\_CACHE\\_MAINT](#) = 3 }
- enum [te\\_ext\\_nv\\_cache\\_maint\\_op\\_t](#) {  
  [OTE\\_EXT\\_NV\\_CM\\_OP\\_CLEAN](#) = 1,  
  [OTE\\_EXT\\_NV\\_CM\\_OP\\_INVALIDATE](#) = 2,  
  [OTE\\_EXT\\_NV\\_CM\\_OP\\_FLUSH](#) = 3 }

### Functions

- [te\\_error\\_t te\\_ext\\_nv\\_cache\\_maint](#) ([te\\_ext\\_nv\\_cache\\_maint\\_op\\_t](#) op, void \*addr, uint32\_t length)
- [te\\_error\\_t te\\_ext\\_nv\\_virt\\_to\\_phys](#) (void \*addr, uint64\_t \*paddr)
- [te\\_error\\_t te\\_ext\\_nv\\_get\\_map\\_addr](#) (uint32\_t id, void \*\*addr)

### 5.3.2 Enumeration Type Documentation

#### 5.3.2.1 anonymous enum

Enumerator:

**[OTE\\_IOCTL\\_GET\\_MAP\\_MEM\\_ADDR](#)**  
**[OTE\\_IOCTL\\_V\\_TO\\_P](#)**  
**[OTE\\_IOCTL\\_CACHE\\_MAINT](#)**

5.3.2.2 enum `te_ext_nv_cache_maint_op_t`

Defines cache maintenance operations.

Enumerator:

- `OTE_EXT_NV_CM_OP_CLEAN`** Cache clean operation.
- `OTE_EXT_NV_CM_OP_INVALIDATE`** Cache invalidate operation.
- `OTE_EXT_NV_CM_OP_FLUSH`** Cache flush operation.

## 5.3.3 Function Documentation

5.3.3.1 `te_error_t te_ext_nv_cache_maint ( te_ext_nv_cache_maint_op_t op, void * addr, uint32_t length )`

Performs a cache maintenance operation.

This API allows the caller to perform a range of platform-specific cache management operations on the specified memory range.

Parameters

in	<i>op</i>	Cache maintenance operation to perform. See <a href="#">te_ext_nv_cache_maint_op_t</a> for more information.
in	<i>addr</i>	Virtual address of the buffer on which the cache maintenance operation is to be performed.
in	<i>length</i>	Length in bytes of the buffer specified by the <i>addr</i> parameter.

Return values

<b><code>OTE_SUCCESS</code></b>	Indicates the operation completed successfully.
<b><code>OTE_ERROR_BAD_PARAMETERS</code></b>	Indicates one or more of the input parameters is invalid: <ul style="list-style-type: none"> <li>The value specified in the ID parameter is not a valid <code>te_ext_nv_cache_maint_op_t</code> value.</li> <li>The virtual address specified in <i>addr</i> is not valid.</li> </ul>
<b><code>OTE_ERROR_OUT_OF_MEMORY</code></b>	Indicates the system ran out of resources.
<b><code>OTE_ERROR_GENERIC</code></b>	Indicates an unspecified error occurred.

5.3.3.2 `te_error_t te_ext_nv_virt_to_phys ( void * addr, uint64_t * paddr )`

Performs virtual-to-physical address translation.

This API returns the physical address that corresponds to the specified virtual address.

Parameters

in	<i>addr</i>	The virtual address to translate.
out	<i>paddr</i>	A pointer with the physical address for <i>addr</i> .

Return values

<b><code>OTE_SUCCESS</code></b>	Indicates the operation completed successfully.
<b><code>OTE_ERROR_BAD_PARAMETERS</code></b>	Indicates one or more of the input parameters is invalid: <ul style="list-style-type: none"> <li>The virtual address specified in <i>addr</i> is not valid.</li> <li>The <i>paddr</i> parameter contains an invalid pointer.</li> </ul>

<i>OTE_ERROR_OUT_OF_MEMORY</i>	Indicates the system ran out of resources.
<i>OTE_ERROR_GENERIC</i>	Indicates an unspecified error occurred.

#### 5.3.3.3 `te_error_t te_ext_nv_get_map_addr ( uint32_t id, void ** addr )`

Retrieves mapping of a specified memory range.

This API returns the mapped address of the specified memory range setup in the service's manifest via the [OTE\\_CONFIG\\_MAP\\_MEM](#) option.

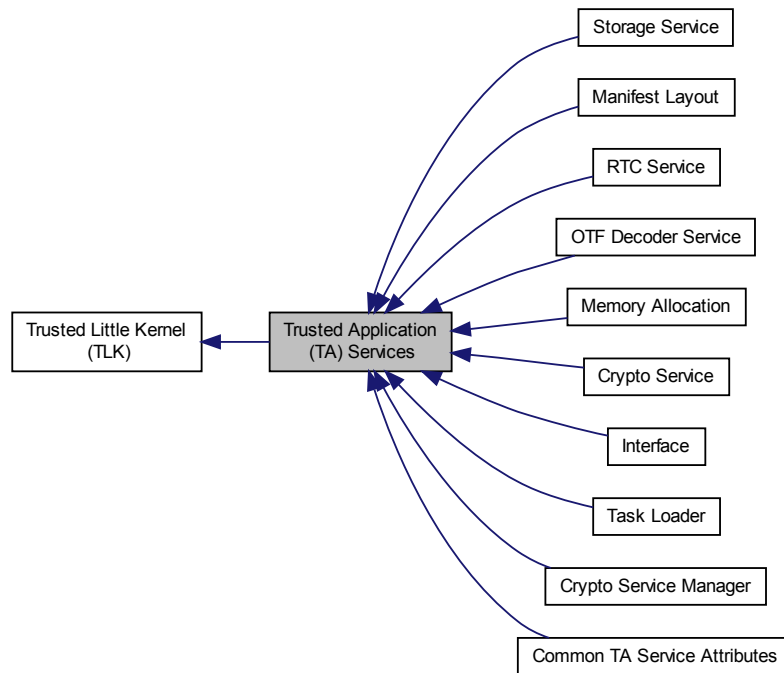
##### Parameters

in	<i>id</i>	Memory region identifier specified in an <code>OTE_CONFIG_MAP_MEM</code> configuration option in the service's manifest.
out	<i>addr</i>	A pointer with the mapped address.

## 5.4 Trusted Application (TA) Services

### 5.4.1 Detailed Description

Declarations and functions for the TA services. Collaboration diagram for Trusted Application (TA) Services:



### Modules

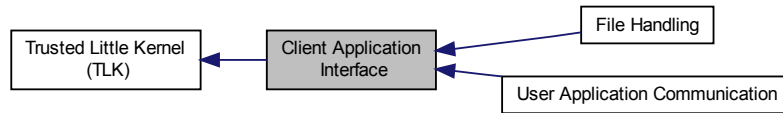
- [Common TA Service Attributes](#)  
*Defines Trusted Little Kernel (TLK) common Trusted Application (TA) service attributes.*
- [Crypto Service](#)  
*Defines APIs for Trusted Little Kernel (TLK) crypto services.*
- [Crypto Service Manager](#)  
*Defines APIs for managing Trusted Little Kernel (TLK) crypto services.*
- [Interface](#)  
*Defines Trusted Application (TA) services declarations and functions.*
- [Manifest Layout](#)  
*Trusted Little Kernel (TLK) services manifest layout.*
- [Memory Allocation](#)  
*Defines Trusted Little Kernel (TLK) memory allocation services functions.*
- [OTF Decoder Service](#)  
*Defines Trusted Little Kernel (TLK) on-the-fly (OTF) decoder services functions.*
- [RTC Service](#)  
*Trusted Little Kernel (TLK) real-time clock (RTC) services.*
- [Storage Service](#)  
*Defines Trusted Little Kernel (TLK) storage services declarations and functions.*
- [Task Loader](#)

*Defines Trusted Application (TA) services declarations and functions for task loading and management.*

## 5.5 Client Application Interface

### 5.5.1 Detailed Description

Defines the client application APIs. Collaboration diagram for Client Application Interface:



### Modules

- [File Handling](#)
- [User Application Communication](#)

### Macros

- `#define TLK_DEVICE_BASE_NAME "tlk_device"`
- `#define TE_IOCTL_MAGIC_NUMBER ('t')`
- `#define TE_IOCTL_OPEN_CLIENT_SESSION _IOWR(TE_IOCTL_MAGIC_NUMBER, 0x10, union te_cmd)`
- `#define TE_IOCTL_CLOSE_CLIENT_SESSION _IOWR(TE_IOCTL_MAGIC_NUMBER, 0x11, union te_cmd)`
- `#define TE_IOCTL_LAUNCH_OP _IOWR(TE_IOCTL_MAGIC_NUMBER, 0x14, union te_cmd)`
- `#define TE_IOCTL_SS_NEW_REQ _IOR(TE_IOCTL_MAGIC_NUMBER, 0x20, ote_ss_op_t)`
- `#define TE_IOCTL_SS_REQ_COMPLETE _IOWR(TE_IOCTL_MAGIC_NUMBER, 0x21, ote_ss_op_t)`

### Enumerations

- `enum {`  
`TLK_SMC_REQUEST = 0xFFFF1000,`  
`TLK_SMC_GET_MORE = 0xFFFF1001,`  
`TLK_SMC_ANSWER = 0xFFFF1002,`  
`TLK_SMC_NO_ANSWER = 0xFFFF1003,`  
`TLK_SMC_OPEN_SESSION = 0xFFFF1004,`  
`TLK_SMC_CLOSE_SESSION = 0xFFFF1005 }`

*Defines secure monitor calls (SMC) that clients use to communicate with trusted applications (TAs) in the secure world.*

### 5.5.2 Macro Definition Documentation

5.5.2.1 `#define TLK_DEVICE_BASE_NAME "tlk_device"`

5.5.2.2 `#define TE_IOCTL_MAGIC_NUMBER ('t')`

5.5.2.3 `#define TE_IOCTL_OPEN_CLIENT_SESSION _IOWR(TE_IOCTL_MAGIC_NUMBER, 0x10, union te_cmd)`

5.5.2.4 `#define TE_IOCTL_CLOSE_CLIENT_SESSION _IOWR(TE_IOCTL_MAGIC_NUMBER, 0x11, union te_cmd)`

5.5.2.5 `#define TE_IOCTL_LAUNCH_OP _IOWR(TE_IOCTL_MAGIC_NUMBER, 0x14, union te_cmd)`

5.5.2.6 `#define TE_IOCTL_SS_NEW_REQ _IOR(TE_IOCTL_MAGIC_NUMBER, 0x20, ote_ss_op_t)`

5.5.2.7 `#define TE_IOCTL_SS_REQ_COMPLETE _IOWR(TE_IOCTL_MAGIC_NUMBER, 0x21, ote_ss_op_t)`

### 5.5.3 Enumeration Type Documentation

#### 5.5.3.1 anonymous enum

Defines secure monitor calls (SMC) that clients use to communicate with trusted applications (TAs) in the secure world.

Enumerator:

***TLK\_SMC\_REQUEST*** Requests OTE to launch a TA operation.

***TLK\_SMC\_GET\_MORE*** Gets a pending answer without making new operation.

***TLK\_SMC\_ANSWER*** Answers from secure side.

***TLK\_SMC\_NO\_ANSWER*** No answers for now (secure side idle).

***TLK\_SMC\_OPEN\_SESSION***

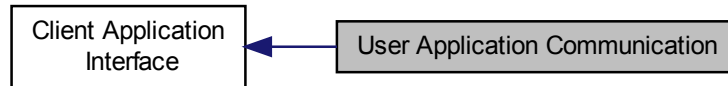
***TLK\_SMC\_CLOSE\_SESSION***



## 5.6 User Application Communication

### 5.6.1 Detailed Description

Collaboration diagram for User Application Communication:



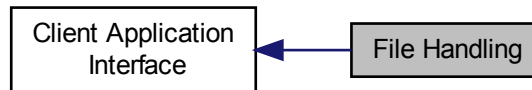
### Data Structures

- struct [te\\_answer](#)
- struct [ote\\_opensession](#)  
*Opens an open trusted environment (OTE) session.*
- struct [ote\\_closesession](#)  
*Closes an OTE session.*
- struct [ote\\_launchop](#)  
*Launches an operation request.*
- union [te\\_cmd](#)

## 5.7 File Handling

### 5.7.1 Detailed Description

Collaboration diagram for File Handling:



### Data Structures

- struct [ote\\_file\\_create\\_params\\_t](#)
- struct [ote\\_file\\_delete\\_params\\_t](#)
- struct [ote\\_file\\_open\\_params\\_t](#)
- struct [ote\\_file\\_close\\_params\\_t](#)
- struct [ote\\_file\\_write\\_params\\_t](#)
- struct [ote\\_file\\_read\\_params\\_t](#)
- struct [ote\\_file\\_seek\\_params\\_t](#)
- struct [ote\\_file\\_trunc\\_params\\_t](#)
- struct [ote\\_file\\_get\\_size\\_params\\_t](#)
- struct [ote\\_file\\_rpmb\\_write\\_params\\_t](#)
- struct [ote\\_file\\_rpmb\\_read\\_params\\_t](#)
- union [ote\\_file\\_req\\_params\\_t](#)
- struct [ote\\_file\\_req\\_t](#)
- struct [ote\\_ss\\_op\\_t](#)

### Macros

- [#define OTE\\_MAX\\_DIR\\_NAME\\_LEN](#) (64)
- [#define OTE\\_MAX\\_FILE\\_NAME\\_LEN](#) (128)
- [#define OTE\\_MAX\\_DATA\\_SIZE](#) (2048)
- [#define OTE\\_RPMB\\_FRAME\\_SIZE](#) 512
- [#define SS\\_OP\\_MAX\\_DATA\\_SIZE](#) 0x1000

### Enumerations

- enum {
  - [OTE\\_FILE\\_REQ\\_TYPE\\_CREATE](#) = 0x1,
  - [OTE\\_FILE\\_REQ\\_TYPE\\_DELETE](#) = 0x2,
  - [OTE\\_FILE\\_REQ\\_TYPE\\_OPEN](#) = 0x3,
  - [OTE\\_FILE\\_REQ\\_TYPE\\_CLOSE](#) = 0x4,
  - [OTE\\_FILE\\_REQ\\_TYPE\\_READ](#) = 0x5,
  - [OTE\\_FILE\\_REQ\\_TYPE\\_WRITE](#) = 0x6,
  - [OTE\\_FILE\\_REQ\\_TYPE\\_GET\\_SIZE](#) = 0x7,
  - [OTE\\_FILE\\_REQ\\_TYPE\\_SEEK](#) = 0x8,
  - [OTE\\_FILE\\_REQ\\_TYPE\\_TRUNC](#) = 0x9,
  - [OTE\\_FILE\\_REQ\\_TYPE\\_RPMB\\_WRITE](#) = 0x1001,

```

    OTE_FILE_REQ_TYPE_RPMB_READ = 0x1002 }
    • enum {
        OTE_FILE_REQ_FLAGS_ACCESS_RO = 1,
        OTE_FILE_REQ_FLAGS_ACCESS_WO = 2,
        OTE_FILE_REQ_FLAGS_ACCESS_RW = 3 }
    • enum {
        OTE_SEEK_WHENCE_SET = 1,
        OTE_SEEK_WHENCE_CUR = 2,
        OTE_SEEK_WHENCE_END = 3 }

```

## 5.7.2 Macro Definition Documentation

5.7.2.1 `#define OTE_MAX_DIR_NAME_LEN (64)`

5.7.2.2 `#define OTE_MAX_FILE_NAME_LEN (128)`

5.7.2.3 `#define OTE_MAX_DATA_SIZE (2048)`

5.7.2.4 `#define OTE_RPMB_FRAME_SIZE 512`

5.7.2.5 `#define SS_OP_MAX_DATA_SIZE 0x1000`

## 5.7.3 Enumeration Type Documentation

5.7.3.1 anonymous enum

Enumerator:

```

    OTE_FILE_REQ_TYPE_CREATE
    OTE_FILE_REQ_TYPE_DELETE
    OTE_FILE_REQ_TYPE_OPEN
    OTE_FILE_REQ_TYPE_CLOSE
    OTE_FILE_REQ_TYPE_READ
    OTE_FILE_REQ_TYPE_WRITE
    OTE_FILE_REQ_TYPE_GET_SIZE
    OTE_FILE_REQ_TYPE_SEEK
    OTE_FILE_REQ_TYPE_TRUNC
    OTE_FILE_REQ_TYPE_RPMB_WRITE
    OTE_FILE_REQ_TYPE_RPMB_READ

```

5.7.3.2 anonymous enum

Enumerator:

```

    OTE_FILE_REQ_FLAGS_ACCESS_RO
    OTE_FILE_REQ_FLAGS_ACCESS_WO
    OTE_FILE_REQ_FLAGS_ACCESS_RW

```

5.7.3.3 anonymous enum

Enumerator:

```

    OTE_SEEK_WHENCE_SET
    OTE_SEEK_WHENCE_CUR
    OTE_SEEK_WHENCE_END

```

## 5.8 Commands

### 5.8.1 Detailed Description

Defines common functions for Trusted Little Kernel (TLK). Collaboration diagram for Commands:



### Functions

- [te\\_error\\_t te\\_open\\_session](#) ([te\\_session\\_t](#) \*session, [te\\_service\\_id\\_t](#) \*service, [te\\_operation\\_t](#) \*operation)
- void [te\\_close\\_session](#) ([te\\_session\\_t](#) \*session)
- [te\\_operation\\_t](#) \* [te\\_create\\_operation](#) (void)
- void [te\\_init\\_operation](#) ([te\\_operation\\_t](#) \*te\_op)
- void [te\\_deinit\\_operation](#) ([te\\_operation\\_t](#) \*teOp)
- [te\\_error\\_t](#) [te\\_launch\\_operation](#) ([te\\_session\\_t](#) \*session, [te\\_operation\\_t](#) \*te\_op)
- void [te\\_oper\\_set\\_command](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t command)
- void [te\\_oper\\_set\\_param\\_int\\_ro](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, uint32\_t Int)
- void [te\\_oper\\_set\\_param\\_int\\_rw](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, uint32\_t Int)
- void [te\\_oper\\_set\\_param\\_mem\\_ro](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, void \*base, uint32\_t len)
- void [te\\_oper\\_set\\_param\\_mem\\_rw](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, void \*base, uint32\_t len)
- uint32\_t [te\\_oper\\_get\\_command](#) ([te\\_operation\\_t](#) \*te\_op)
- uint32\_t [te\\_oper\\_get\\_num\\_params](#) ([te\\_operation\\_t](#) \*te\_op)
- [te\\_error\\_t](#) [te\\_oper\\_get\\_param\\_type](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, [te\\_oper\\_param\\_type\\_t](#) \*type)
- [te\\_error\\_t](#) [te\\_oper\\_get\\_param\\_int](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, uint32\_t \*Int)
- [te\\_error\\_t](#) [te\\_oper\\_get\\_param\\_mem](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, void \*\*base, uint32\_t \*len)
- void [te\\_operation\\_reset](#) ([te\\_operation\\_t](#) \*te\_op)

### 5.8.2 Function Documentation

#### 5.8.2.1 [te\\_error\\_t te\\_open\\_session](#) ( [te\\_session\\_t](#) \* session, [te\\_service\\_id\\_t](#) \* service, [te\\_operation\\_t](#) \* operation )

Opens a session to a TLK secure service.

Creates a session to a specified TLK secure service. This function must be called from the user side or from another TLK service.

#### Parameters

out	<i>session</i>	A pointer to a TLK secure service session to be initialized by this function.
in	<i>service</i>	The UUID/Service ID of the TLK service to which the caller wants to connect.
in, out	<i>operation</i>	A pointer to parameters that are expected by the TLK service.

#### Returns

OTE\_SUCCESS Indicates the operation was successful.

**5.8.2.2 void te\_close\_session ( te\_session\_t \* session )**

Closes an existing open session to a TLK secure service.

**Parameters**

in	<i>session</i>	A pointer to a TLK secure service session.
----	----------------	--

**5.8.2.3 te\_operation\_t\* te\_create\_operation ( void )**

Dynamically creates a new TLK secure service operation object.

This function creates a new `te_operation_t` object on the heap, and then initializes it.

**Returns**

A pointer to a TLK secure service operation object that was allocated on the stack and initialized.

**5.8.2.4 void te\_init\_operation ( te\_operation\_t \* te\_op )**

Initializes a TLK operation object.

This function initializes a TLK operation object using the TLK operation object pointer parameter passed to it. It is normally called to allocate a TLK operation object on the stack.

**Parameters**

in, out	<i>te_op</i>	A pointer to a TLK operation object.
---------	--------------	--------------------------------------

**5.8.2.5 void te\_deinit\_operation ( te\_operation\_t \* teOp )**

Deinitializes an existing TLK operation object.

Frees internal memory used by an existing TLK operation object.

**Parameters**

in	<i>teOp</i>	A pointer to a TLK secure service operation object.
----	-------------	---

**5.8.2.6 te\_error\_t te\_launch\_operation ( te\_session\_t \* session, te\_operation\_t \* te\_op )**

Sends an existing TLK operation object.

Sends the operation object to the TLK service. The operation object must have the command ID or necessary parameter setup.

**Parameters**

in	<i>session</i>	A pointer to an established TLK service session.
in, out	<i>te_op</i>	A <code>te_operation_t</code> object with proper parameters set up and a command ID.

**Returns**

The TLK secure service results after it services the command.

### 5.8.2.7 void te\_oper\_set\_command ( te\_operation\_t \* te\_op, uint32\_t command )

Sets a command to a TLK secure service operation object.

Sets the command ID to the operation object that will be sent to the TLK secure service.

#### Parameters

in	<i>te_op</i>	A pointer to a TLK operation object.
in	<i>command</i>	A command ID that will be sent to the TLK secure service.

### 5.8.2.8 void te\_oper\_set\_param\_int\_ro ( te\_operation\_t \* te\_op, uint32\_t index, uint32\_t Int )

Adds a read-only integer parameter to a TLK operation object.

#### Parameters

in	<i>te_op</i>	A pointer to a TLK operation object.
in	<i>index</i>	An index value used to retrieve the parameter.
in	<i>Int</i>	An integer value.

### 5.8.2.9 void te\_oper\_set\_param\_int\_rw ( te\_operation\_t \* te\_op, uint32\_t index, uint32\_t Int )

Adds a read-write integer parameter to a TLK operation object.

#### Parameters

in	<i>te_op</i>	A pointer to a TLK operation object.
in	<i>index</i>	An index value used to retrieve the parameter.
in	<i>Int</i>	An integer value.

### 5.8.2.10 void te\_oper\_set\_param\_mem\_ro ( te\_operation\_t \* te\_op, uint32\_t index, void \* base, uint32\_t len )

Adds a read-only buffer parameter to the operation object. The buffer is the parameter at the given *index* in the object.

#### Parameters

in	<i>te_op</i>	A pointer to a TLK operation object.
in	<i>index</i>	An index value used to retrieve the parameter.
in	<i>base</i>	The base address to the memory buffer.
in	<i>len</i>	The length of the memory buffer.

### 5.8.2.11 void te\_oper\_set\_param\_mem\_rw ( te\_operation\_t \* te\_op, uint32\_t index, void \* base, uint32\_t len )

Adds a read-write buffer parameter to the operation object. The buffer is the parameter at the given *index* in the object.

#### Parameters

in	<i>te_op</i>	A pointer to a TLK operation object.
in	<i>index</i>	An index value used to retrieve the parameter.
in	<i>base</i>	The base address to the memory buffer.
in	<i>len</i>	The length of the memory buffer.

5.8.2.12 `uint32_t te_oper_get_command ( te_operation_t * te_op )`

Gets a TLK command from an operation object.

## Parameters

in	<i>te_op</i>	A pointer to a TLK operation object.
----	--------------	--------------------------------------

## Returns

The current command ID from the given operation object.

5.8.2.13 `uint32_t te_oper_get_num_params ( te_operation_t * te_op )`

Gets the number of parameters from an operation object.

## Parameters

in	<i>te_op</i>	A pointer to a TLK operation object.
----	--------------	--------------------------------------

## Returns

The number of parameters from the given operation object.

5.8.2.14 `te_error_t te_oper_get_param_type ( te_operation_t * te_op, uint32_t index, te_oper_param_type_t * type )`

Gets the parameter type of a parameter.

Gets the parameter type of a parameter, given its index and operation object.

## Parameters

in	<i>te_op</i>	A pointer to a TLK operation object.
in	<i>index</i>	An index value used to retrieve the parameter.
out	<i>type</i>	The type of parameter.

## Return values

<i>OTE_SUCCESS</i>	Indicates the <i>te_op</i> parameter was found and the <i>type</i> parameter was returned.
<i>OTE_ERROR_ITEM_NOT_FOUND</i>	Indicates the <i>te_op</i> parameter was not found or the <i>type</i> parameter was not returned.

5.8.2.15 `te_error_t te_oper_get_param_int ( te_operation_t * te_op, uint32_t index, uint32_t * Int )`

Gets an integer parameter from a given TLK operation object.

## Parameters

in	<i>te_op</i>	A pointer to a TLK operation object.
in	<i>index</i>	An index value used to retrieve the parameter.
out	<i>Int</i>	A pointer to an integer value.

## Return values

<i>OTE_SUCCESS</i>	Indicates the <i>te_op</i> parameter was found and the <i>Int</i> parameter was returned.
<i>OTE_ERROR_ITEM_NOT_FOUND</i>	Indicates the <i>index</i> was not found or the <i>Int</i> parameter was not an integer.

5.8.2.16 `te_error_t te_oper_get_param_mem ( te_operation_t * te_op, uint32_t index, void ** base, uint32_t * len )`

Gets a memory buffer parameter from a given TLK operation object.

## Parameters

in	<i>te_op</i>	A pointer to a TLK operation object.
in	<i>index</i>	An index value used to retrieve the parameter.
out	<i>base</i>	The base address of buffer.
out	<i>len</i>	The length of memory buffer.

## Return values

<i>OTE_SUCCESS</i>	Indicates the <i>te_op</i> parameter was found and the <i>index</i> was returned.
<i>OTE_ERROR_ITEM_NOT_FOUND</i>	Indicates the <i>index</i> was not found or the <i>te_op</i> parameter was not a memory buffer.

5.8.2.17 `void te_operation_reset ( te_operation_t * te_op )`

Resets the data in an operation object.

When you call functions that set values within an operation object (for example [te\\_oper\\_set\\_param\\_mem\\_rw\(\)](#)), the TLK interface allocates internal memory for those values. If you need to re-use an operation with new values, call `te_operation_reset()` before setting the new values.

## Parameters

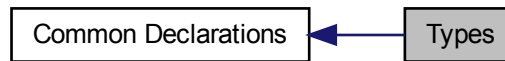
in	<i>te_op</i>	A pointer to a TLK operation object.
----	--------------	--------------------------------------



## 5.9 Types

### 5.9.1 Detailed Description

Defines common data types and functions for Trusted Little Kernel (TLK). Collaboration diagram for Types:



#### Data Structures

- struct [te\\_service\\_id\\_t](#)
- union [te\\_session\\_t](#)
- struct [te\\_oper\\_param\\_t](#)
- struct [te\\_operation\\_t](#)

#### Macros

- `#define` [OTE\\_TASK\\_NAME\\_MAX\\_LENGTH](#) 24
- `#define` [OTE\\_TASK\\_PRIVATE\\_DATA\\_LENGTH](#) 20

#### Typedefs

- typedef uint64\_t [cmnptr\\_t](#)

#### Enumerations

- enum [te\\_oper\\_param\\_type\\_t](#) {  
[TE\\_PARAM\\_TYPE\\_NONE](#) = 0,  
[TE\\_PARAM\\_TYPE\\_INT\\_RO](#) = 1,  
[TE\\_PARAM\\_TYPE\\_INT\\_RW](#) = 2,  
[TE\\_PARAM\\_TYPE\\_MEM\\_RO](#) = 3,  
[TE\\_PARAM\\_TYPE\\_MEM\\_RW](#) = 4 }

#### Functions

- [te\\_result\\_origin\\_t](#) [te\\_get\\_result\\_origin](#) ([te\\_session\\_t](#) \*session)

### 5.9.2 Macro Definition Documentation

#### 5.9.2.1 `#define` [OTE\\_TASK\\_NAME\\_MAX\\_LENGTH](#) 24

Defines the maximum length of a zero-terminated informative task name.

### 5.9.2.2 `#define OTE_TASK_PRIVATE_DATA_LENGTH 20`

Defines the length of private data for the Trusted Application (TA). The definition goes in the manifest. The semantics of this optional data is defined per each TA.

To hold an SHA1 digest, this definition must be at least 20 bytes. Such a digest enables building a chain of trust to a TA with the manifest data.

This definition can be used to perform tasks such as:

- Loading public keys
- Loading X509 certificates and digests

## 5.9.3 Typedef Documentation

### 5.9.3.1 `typedef uint64_t cmnptr_t`

Holds a pointer large enough to support 32- and 64-bit clients.

## 5.9.4 Enumeration Type Documentation

### 5.9.4.1 `enum te_oper_param_type_t`

Specifies the operation object's parameter types.

Enumerator:

```
TE_PARAM_TYPE_NONE
TE_PARAM_TYPE_INT_RO
TE_PARAM_TYPE_INT_RW
TE_PARAM_TYPE_MEM_RO
TE_PARAM_TYPE_MEM_RW
```

## 5.9.5 Function Documentation

### 5.9.5.1 `te_result_origin_t te_get_result_origin ( te_session_t * session )`

Returns the origin of a returned result.

Because it is possible for the operation to fail anywhere in the pipeline, this function returns the general block where the returned result originated.

Parameters

<code>in</code>	<code>session</code>	A valid session pointer.
-----------------	----------------------	--------------------------

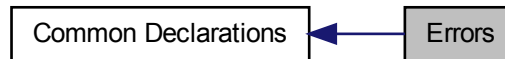
Returns

A `te_result_origin_t` ID number.

## 5.10 Errors

### 5.10.1 Detailed Description

Defines common error codes for Trusted Little Kernel (TLK). Collaboration diagram for Errors:



### Enumerations

- enum `te_error_t` {
  - `OTE_SUCCESS` = 0x00000000,
  - `OTE_ERROR_NO_ERROR` = 0x00000000,
  - `OTE_ERROR_GENERIC` = 0xFFFF0000,
  - `OTE_ERROR_ACCESS_DENIED` = 0xFFFF0001,
  - `OTE_ERROR_CANCEL` = 0xFFFF0002,
  - `OTE_ERROR_ACCESS_CONFLICT` = 0xFFFF0003,
  - `OTE_ERROR_EXCESS_DATA` = 0xFFFF0004,
  - `OTE_ERROR_BAD_FORMAT` = 0xFFFF0005,
  - `OTE_ERROR_BAD_PARAMETERS` = 0xFFFF0006,
  - `OTE_ERROR_BAD_STATE` = 0xFFFF0007,
  - `OTE_ERROR_ITEM_NOT_FOUND` = 0xFFFF0008,
  - `OTE_ERROR_NOT_IMPLEMENTED` = 0xFFFF0009,
  - `OTE_ERROR_NOT_SUPPORTED` = 0xFFFF000A,
  - `OTE_ERROR_NO_DATA` = 0xFFFF000B,
  - `OTE_ERROR_OUT_OF_MEMORY` = 0xFFFF000C,
  - `OTE_ERROR_BUSY` = 0xFFFF000D,
  - `OTE_ERROR_COMMUNICATION` = 0xFFFF000E,
  - `OTE_ERROR_SECURITY` = 0xFFFF000F,
  - `OTE_ERROR_SHORT_BUFFER` = 0xFFFF0010,
  - `OTE_ERROR_BLOCKED` = 0xFFFF0011,
  - `OTE_ERROR_NO_ANSWER` = 0xFFFF1003 }

*Defines Open Trusted Environment (OTE) error codes.*

- enum `te_result_origin_t` {
  - `OTE_RESULT_ORIGIN_API` = 1,
  - `OTE_RESULT_ORIGIN_COMMS` = 2,
  - `OTE_RESULT_ORIGIN_KERNEL` = 3,
  - `OTE_RESULT_ORIGIN_TRUSTED_APP` = 4 }

*Defines the origin of an error.*

### 5.10.2 Enumeration Type Documentation

#### 5.10.2.1 enum `te_error_t`

Defines Open Trusted Environment (OTE) error codes.

Enumerator:

- OTE\_SUCCESS** Indicates the operation was successful.
- OTE\_ERROR\_NO\_ERROR** Indicates the operation was successful.
- OTE\_ERROR\_GENERIC** Indicates an unspecified error occurred.
- OTE\_ERROR\_ACCESS\_DENIED** Indicates access privileges are insufficient.
- OTE\_ERROR\_CANCEL** Indicates the operation was cancelled.
- OTE\_ERROR\_ACCESS\_CONFLICT** Indicates a concurrent accesses conflict.
- OTE\_ERROR\_EXCESS\_DATA** Indicates data passed exceeds request.
- OTE\_ERROR\_BAD\_FORMAT** Indicates input data is in an invalid format.
- OTE\_ERROR\_BAD\_PARAMETERS** Indicates input parameters are invalid.
- OTE\_ERROR\_BAD\_STATE** Indicates the operation is invalid in its current state.
- OTE\_ERROR\_ITEM\_NOT\_FOUND** Indicates the requested data item was not found.
- OTE\_ERROR\_NOT\_IMPLEMENTED** Indicates the requested operation was not implemented.
- OTE\_ERROR\_NOT\_SUPPORTED** Indicates the requested operation is not supported.
- OTE\_ERROR\_NO\_DATA** Indicates the data expected is missing.
- OTE\_ERROR\_OUT\_OF\_MEMORY** Indicates the system ran out of resources.
- OTE\_ERROR\_BUSY** Indicates the system is busy.
- OTE\_ERROR\_COMMUNICATION** Indicates that communication failed.
- OTE\_ERROR\_SECURITY** Indicates a security fault was detected.
- OTE\_ERROR\_SHORT\_BUFFER** Indicates the supplied buffer is too short.
- OTE\_ERROR\_BLOCKED** Task administratively blocked, does not accept new sessions.
- OTE\_ERROR\_NO\_ANSWER** Indicates no answer was received from the command target.

#### 5.10.2.2 enum te\_result\_origin\_t

Defines the origin of an error.

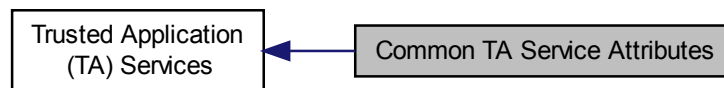
Enumerator:

- OTE\_RESULT\_ORIGIN\_API** Indicates the error originated from the OTE Client API.
- OTE\_RESULT\_ORIGIN\_COMMS** Indicates the error originated from the underlying communications stack.
- OTE\_RESULT\_ORIGIN\_KERNEL** Indicates the error originated from the common OTE kernel code.
- OTE\_RESULT\_ORIGIN\_TRUSTED\_APP** Indicates the error originated from the Trusted Application code.

## 5.11 Common TA Service Attributes

### 5.11.1 Detailed Description

Defines Trusted Little Kernel (TLK) common Trusted Application (TA) service attributes. Collaboration diagram for Common TA Service Attributes:



### Data Structures

- struct [te\\_attribute\\_t](#)

### Macros

- `#define OTE_ATTR_VAL 1 << 29`
- `#define OTE_ATTR_PUB 1 << 28`

### Enumerations

- enum [te\\_attribute\\_id\\_t](#) {  
[OTE\\_ATTR\\_SECRET\\_VALUE](#) = 0xC0000000,  
[OTE\\_ATTR\\_RSA\\_MODULES](#) = 0xD0000130,  
[OTE\\_ATTR\\_RSA\\_PUBLIC\\_EXPONENT](#) = 0xD0000230,  
[OTE\\_ATTR\\_RSA\\_PRIVATE\\_EXPONENT](#) = 0xC0000330,  
[OTE\\_ATTR\\_RSA\\_PRIME1](#) = 0xC0000430,  
[OTE\\_ATTR\\_RSA\\_PRIME2](#) = 0xC0000530,  
[OTE\\_ATTR\\_RSA\\_EXPONENT1](#) = 0xC0000630,  
[OTE\\_ATTR\\_RSA\\_EXPONENT2](#) = 0xC0000730,  
[OTE\\_ATTR\\_RSA\\_COEFFICIENT](#) = 0xC0000830,  
[OTE\\_ATTR\\_DSA\\_PRIME](#) = 0xD0001031,  
[OTE\\_ATTR\\_DSA\\_SUBPRIME](#) = 0xD0001131,  
[OTE\\_ATTR\\_DSA\\_BASE](#) = 0xD0001231,  
[OTE\\_ATTR\\_DSA\\_PUBLIC\\_VALUE](#) = 0xD0000131,  
[OTE\\_ATTR\\_DSA\\_PRIVATE\\_VALUE](#) = 0xD0000231,  
[OTE\\_ATTR\\_DH\\_PRIME](#) = 0xD0001032,  
[OTE\\_ATTR\\_DH\\_SUBPRIME](#) = 0xD0001132,  
[OTE\\_ATTR\\_DH\\_BASE](#) = 0xD0001232,  
[OTE\\_ATTR\\_DH\\_X\\_BITS](#) = 0xF0001332,  
[OTE\\_ATTR\\_DH\\_PUBLIC\\_VALUE](#) = 0xD0000132,  
[OTE\\_ATTR\\_DH\\_PRIVATE\\_VALUE](#) = 0xC0000232,  
[OTE\\_ATTR\\_RSA\\_OAEP\\_LABEL](#) = 0xD0000930,  
[OTE\\_ATTR\\_RSA\\_PSS\\_SALT\\_LENGTH](#) = 0xF0000A30 }

## Functions

- [te\\_error\\_t te\\_set\\_mem\\_attribute](#) ([te\\_attribute\\_t](#) \*attr, [te\\_attribute\\_id\\_t](#) id, void \*buffer, [uint32\\_t](#) size)
- [te\\_error\\_t te\\_get\\_mem\\_attribute\\_buffer](#) ([te\\_attribute\\_t](#) \*attr, void \*\*ret)
- [te\\_error\\_t te\\_get\\_mem\\_attribute\\_size](#) ([te\\_attribute\\_t](#) \*attr, [size\\_t](#) \*ret)
- void [te\\_copy\\_mem\\_attribute](#) (void \*buffer, [te\\_attribute\\_t](#) \*key)
- [te\\_error\\_t te\\_set\\_int\\_attribute](#) ([te\\_attribute\\_t](#) \*attr, [te\\_attribute\\_id\\_t](#) id, [uint32\\_t](#) a, [uint32\\_t](#) b)
- void [te\\_free\\_internal\\_attribute](#) ([te\\_attribute\\_t](#) \*attr)
- [te\\_error\\_t te\\_copy\\_attribute](#) ([te\\_attribute\\_t](#) \*dst, [te\\_attribute\\_t](#) \*src)

### 5.11.2 Macro Definition Documentation

#### 5.11.2.1 #define OTE\_ATTR\_VAL 1 << 29

Defines attribute ID type flag bitmasks.

#### 5.11.2.2 #define OTE\_ATTR\_PUB 1 << 28

### 5.11.3 Enumeration Type Documentation

#### 5.11.3.1 enum te\_attribute\_id\_t

Defines attribute ID types.

Enumerator:

```

OTE_ATTR_SECRET_VALUE
OTE_ATTR_RSA_MODULES OTE_ATTR_PUB.
OTE_ATTR_RSA_PUBLIC_EXPONENT OTE_ATTR_PUB.
OTE_ATTR_RSA_PRIVATE_EXPONENT
OTE_ATTR_RSA_PRIME1
OTE_ATTR_RSA_PRIME2
OTE_ATTR_RSA_EXPONENT1
OTE_ATTR_RSA_EXPONENT2
OTE_ATTR_RSA_COEFFICIENT
OTE_ATTR_DSA_PRIME
OTE_ATTR_DSA_SUBPRIME OTE_ATTR_PUB.
OTE_ATTR_DSA_BASE OTE_ATTR_PUB.
OTE_ATTR_DSA_PUBLIC_VALUE OTE_ATTR_PUB.
OTE_ATTR_DSA_PRIVATE_VALUE OTE_ATTR_PUB.
OTE_ATTR_DH_PRIME OTE_ATTR_PUB.
OTE_ATTR_DH_SUBPRIME OTE_ATTR_PUB.
OTE_ATTR_DH_BASE OTE_ATTR_PUB.
OTE_ATTR_DH_X_BITS OTE_ATTR_VAL | OTE_ATTR_PUB.
OTE_ATTR_DH_PUBLIC_VALUE OTE_ATTR_PUB.
OTE_ATTR_DH_PRIVATE_VALUE
OTE_ATTR_RSA_OAEP_LABEL OTE_ATTR_PUB.
OTE_ATTR_RSA_PSS_SALT_LENGTH OTE_ATTR_VAL | OTE_ATTR_PUB.

```

### 5.11.4 Function Documentation

#### 5.11.4.1 `te_error_t te_set_mem_attribute ( te_attribute_t * attr, te_attribute_id_t id, void * buffer, uint32_t size )`

Sets up a memory attribute struct for use in other functions. This function stores the *id*, *buffer*, and *size* parameters in the memory attribute. The *id* parameter must be compatible with the memory attributes.

See Also

[te\\_get\\_mem\\_attribute\\_buffer\(\)](#) and [te\\_get\\_mem\\_attribute\\_size\(\)](#)

#### Parameters

in, out	<i>attr</i>	Attribute object.
in	<i>id</i>	Attribute ID of the buffer.
in	<i>buffer</i>	Memory location of the buffer.
in	<i>size</i>	Size of the buffer.

#### Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

#### 5.11.4.2 `te_error_t te_get_mem_attribute_buffer ( te_attribute_t * attr, void ** ret )`

Gets a memory attribute buffer. Stores in *ret* the address of the buffer in the memory attribute.

#### Parameters

in	<i>attr</i>	Attribute object.
out	<i>ret</i>	Return buffer pointer.

#### Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

#### 5.11.4.3 `te_error_t te_get_mem_attribute_size ( te_attribute_t * attr, size_t * ret )`

Gets memory attribute size. Stores in *ret* the size of the buffer in the mem attribute.

#### Parameters

in	<i>attr</i>	Attribute object.
out	<i>ret</i>	Return size.

#### Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

#### 5.11.4.4 `void te_copy_mem_attribute ( void * buffer, te_attribute_t * key )`

Copies the memory attribute. Copies the memory attribute buffer to the destination buffer, which must previously be allocated.

## Parameters

<i>in, out</i>	<i>buffer</i>	Destination buffer.
<i>in</i>	<i>key</i>	Source attribute.

5.11.4.5 `te_error_t te_set_int_attribute ( te_attribute_t * attr, te_attribute_id_t id, uint32_t a, uint32_t b )`

Sets the integer attribute. Sets the attribute with integer values and *id*. The *id* parameter must match the integer type.

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

## Parameters

<i>in, out</i>	<i>attr</i>	Attribute object.
<i>in</i>	<i>id</i>	Attribute ID.
<i>in</i>	<i>a</i>	Integer value.
<i>in</i>	<i>b</i>	Integer value.

5.11.4.6 `void te_free_internal_attribute ( te_attribute_t * attr )`

Frees internal attribute memory. Frees any memory references by attribute.

## Parameters

<i>in</i>	<i>attr</i>	Attribute object.
-----------	-------------	-------------------

5.11.4.7 `te_error_t te_copy_attribute ( te_attribute_t * dst, te_attribute_t * src )`

Copies attribute internals. After allocating space in destination attribute, this function copies integer memory attributes.

## Parameters

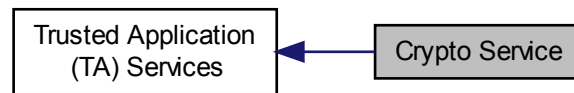
<i>in, out</i>	<i>dst</i>	Destination attribute.
<i>in</i>	<i>src</i>	Source attribute.



## 5.12 Crypto Service

### 5.12.1 Detailed Description

Defines APIs for Trusted Little Kernel (TLK) crypto services. Collaboration diagram for Crypto Service:



### Data Structures

- struct [te\\_crypto\\_operation\\_info\\_t](#)
- struct [\\_\\_te\\_crypto\\_operation\\_t](#)
- struct [te\\_crypto\\_rsa\\_key\\_t](#)

### Typedefs

- typedef struct [\\_\\_te\\_crypto\\_object](#) \* [te\\_crypto\\_object\\_t](#)
- typedef struct [\\_\\_te\\_crypto\\_operation\\_t](#) \* [te\\_crypto\\_operation\\_t](#)

### Enumerations

- enum [te\\_oper\\_crypto\\_algo\\_t](#) {  
[OTE\\_ALG\\_AES\\_ECB\\_NOPAD](#) = 0x10000010,  
[OTE\\_ALG\\_AES\\_CBC\\_NOPAD](#) = 0x10000110,  
[OTE\\_ALG\\_AES\\_CTR](#) = 0x10000210,  
[OTE\\_ALG\\_AES\\_CTS](#) = 0x10000310,  
[OTE\\_ALG\\_AES\\_ECB](#) = 0x10000510,  
[OTE\\_ALG\\_AES\\_CBC](#) = 0x10000610,  
[OTE\\_ALG\\_AES\\_CMAC\\_128](#) = 0x20000110,  
[OTE\\_ALG\\_AES\\_CMAC\\_192](#) = 0x20000120,  
[OTE\\_ALG\\_AES\\_CMAC\\_256](#) = 0x20000130,  
[OTE\\_ALG\\_SHA\\_HMAC\\_224](#) = 0x20000210,  
[OTE\\_ALG\\_SHA\\_HMAC\\_256](#) = 0x20000220,  
[OTE\\_ALG\\_SHA\\_HMAC\\_384](#) = 0x20000230,  
[OTE\\_ALG\\_SHA\\_HMAC\\_512](#) = 0x20000240,  
[OTE\\_ALG\\_RSA\\_PKCS\\_OAEP](#) = 0x30000100,  
[OTE\\_ALG\\_RSA\\_PSS](#) = 0x30000200 }
- enum [te\\_oper\\_crypto\\_algo\\_mode\\_t](#) {  
[OTE\\_ALG\\_MODE\\_ENCRYPT](#),  
[OTE\\_ALG\\_MODE\\_DECRYPT](#),  
[OTE\\_ALG\\_MODE\\_SIGN](#),  
[OTE\\_ALG\\_MODE\\_VERIFY](#),  
[OTE\\_ALG\\_MODE\\_DIGEST](#),  
[OTE\\_ALG\\_MODE\\_DERIVE](#) }

## Functions

- [te\\_error\\_t te\\_allocate\\_object](#) ([te\\_crypto\\_object\\_t](#) \*obj)
- [te\\_error\\_t te\\_populate\\_object](#) ([te\\_crypto\\_object\\_t](#) obj, [te\\_attribute\\_t](#) \*attrs, [uint32\\_t](#) attr\_count)
- [void te\\_free\\_object](#) ([te\\_crypto\\_object\\_t](#) obj)
- [te\\_error\\_t te\\_allocate\\_operation](#) ([te\\_crypto\\_operation\\_t](#) \*oper, [te\\_oper\\_crypto\\_algo\\_t](#) algorithm, [te\\_oper\\_crypto\\_algo\\_mode\\_t](#) mode)
- [te\\_error\\_t te\\_set\\_operation\\_key](#) ([te\\_crypto\\_operation\\_t](#) oper, [te\\_crypto\\_object\\_t](#) obj)
- [te\\_error\\_t te\\_cipher\\_init](#) ([te\\_crypto\\_operation\\_t](#) oper, void \*iv, [uint32\\_t](#) iv\_size)
- [te\\_error\\_t te\\_cipher\\_update](#) ([te\\_crypto\\_operation\\_t](#) oper, void \*src\_data, [uint32\\_t](#) src\_size, void \*dst\_data, [uint32\\_t](#) \*dst\_size)
- [te\\_error\\_t te\\_cipher\\_do\\_final](#) ([te\\_crypto\\_operation\\_t](#) oper, void \*src\_data, [uint32\\_t](#) src\_len, void \*dst\_data, [uint32\\_t](#) \*dst\_len)
- [te\\_error\\_t te\\_rsa\\_init](#) ([te\\_crypto\\_operation\\_t](#) oper)
- [te\\_error\\_t te\\_rsa\\_handle\\_request](#) ([te\\_crypto\\_operation\\_t](#) oper, void \*src\_data, [uint32\\_t](#) src\_size, void \*dst\_data, [uint32\\_t](#) \*dst\_size)
- [void te\\_free\\_operation](#) ([te\\_crypto\\_operation\\_t](#) oper)
- [void te\\_generate\\_random](#) (void \*buffer, [size\\_t](#) size)
- [te\\_error\\_t te\\_get\\_attribute\\_by\\_id](#) ([te\\_crypto\\_object\\_t](#) object, [te\\_attribute\\_id\\_t](#) id, [te\\_attribute\\_t](#) \*\*ret)

## 5.12.2 Typedef Documentation

5.12.2.1 `typedef struct __te_crypto_object* te_crypto_object_t`

5.12.2.2 `typedef struct __te_crypto_operation_t* te_crypto_operation_t`

## 5.12.3 Enumeration Type Documentation

5.12.3.1 `enum te_oper_crypto_algo_t`

Defines algorithm types.

Enumerator:

```

OTE_ALG_AES_ECB_NOPAD
OTE_ALG_AES_CBC_NOPAD
OTE_ALG_AES_CTR
OTE_ALG_AES_CTS
OTE_ALG_AES_ECB
OTE_ALG_AES_CBC
OTE_ALG_AES_CMAC_128
OTE_ALG_AES_CMAC_192
OTE_ALG_AES_CMAC_256
OTE_ALG_SHA_HMAC_224
OTE_ALG_SHA_HMAC_256
OTE_ALG_SHA_HMAC_384
OTE_ALG_SHA_HMAC_512
OTE_ALG_RSA_PKCS_OAEP
OTE_ALG_RSA_PSS

```

## 5.12.3.2 enum te\_oper\_crypto\_algo\_mode\_t

Defines algorithm modes.

Enumerator:

**OTE\_ALG\_MODE\_ENCRYPT**  
**OTE\_ALG\_MODE\_DECRYPT**  
**OTE\_ALG\_MODE\_SIGN**  
**OTE\_ALG\_MODE\_VERIFY**  
**OTE\_ALG\_MODE\_DIGEST**  
**OTE\_ALG\_MODE\_DERIVE**

## 5.12.4 Function Documentation

## 5.12.4.1 te\_error\_t te\_allocate\_object ( te\_crypto\_object\_t \* obj )

Allocates memory for a [te\\_crypto\\_object\\_t](#).

Parameters

in, out	obj	A pointer to new crypto object.
---------	-----	---------------------------------

Return values

OTE_SUCCESS	Indicates the operation was successful.
OTE_ERROR_OUT_OF_MEMORY	Indicates the system ran out of resources.

## 5.12.4.2 te\_error\_t te\_populate\_object ( te\_crypto\_object\_t obj, te\_attribute\_t \* attrs, uint32\_t attr\_count )

Populates crypto object from a list of attributes. Allocates *obj* internal memory and copies attributes to *obj*.

Parameters

in, out	obj	Crypto object to store attributes.
in	attrs	Array of attributes.
in	attr_count	Array length.

Return values

OTE_SUCCESS	Indicates the operation was successful.
-------------	---

## 5.12.4.3 void te\_free\_object ( te\_crypto\_object\_t obj )

Frees allocated memory within crypto object.

Parameters

in	obj	Crypto object.
----	-----	----------------

#### 5.12.4.4 `te_error_t te_allocate_operation ( te_crypto_operation_t * oper, te_oper_crypto_algo_t algorithm, te_oper_crypto_algo_mode_t mode )`

Allocates memory for crypto operation. Allocates crypto operation internal memory. Initializes operation based on algo/mode.

##### Parameters

in, out	<i>oper</i>	Crypto operation object.
in	<i>algorithm</i>	Crypto algorithm.
in	<i>mode</i>	Crypto algorithm mode.

##### Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

#### 5.12.4.5 `te_error_t te_set_operation_key ( te_crypto_operation_t oper, te_crypto_object_t obj )`

Allocates memory in the crypto operation and copies the key from the crypto object to the operation object.

##### Parameters

in, out	<i>oper</i>	Crypto operation object.
in	<i>obj</i>	Key crypto object.

##### Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

#### 5.12.4.6 `te_error_t te_cipher_init ( te_crypto_operation_t oper, void * iv, uint32_t iv_size )`

Initializes the operation cipher. Sets the initialization vector and calls the `init` function.

##### Parameters

in, out	<i>oper</i>	Crypto operation object.
in	<i>iv</i>	Initialization vector.
in	<i>iv_size</i>	Initialization vector size.

##### Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

#### 5.12.4.7 `te_error_t te_cipher_update ( te_crypto_operation_t oper, void * src_data, uint32_t src_size, void * dst_data, uint32_t * dst_size )`

Updates the cipher by calling the operation `update` function with the supplied parameters.

##### Parameters

in	<i>oper</i>	Crypto operation object
in	<i>src_data</i>	Source data buffer supplied to <code>init</code> .
in	<i>src_size</i>	Source buffer size supplied to <code>init</code> .
in, out	<i>dst_data</i>	Destination data buffer supplied to <code>init</code> .
in, out	<i>dst_size</i>	Destination buffer size supplied to <code>init</code> .

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

**5.12.4.8** `te_error_t te_cipher.do_final ( te_crypto_operation_t oper, void * src_data, uint32_t src_len, void * dst_data, uint32_t * dst_len )`

Calls operation `do_final` with supplied parameters.

## Parameters

in	<i>oper</i>	Crypto operation object.
in	<i>src_data</i>	Source data buffer supplied to <code>do_final</code> .
in	<i>src_len</i>	Source buffer size supplied to <code>do_final</code> .
in, out	<i>dst_data</i>	Destination data buffer supplied to <code>do_final</code> .
in, out	<i>dst_len</i>	Destination buffer size supplied to <code>do_final</code> .

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

**5.12.4.9** `te_error_t te_rsa.init ( te_crypto_operation_t oper )`

Initializes the RSA operation.

## Parameters

in, out	<i>oper</i>	Crypto operation object.
---------	-------------	--------------------------

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

**5.12.4.10** `te_error_t te_rsa.handle_request ( te_crypto_operation_t oper, void * src_data, uint32_t src_size, void * dst_data, uint32_t * dst_size )`

Executes RSA operations.

## Parameters

in	<i>oper</i>	Crypto operation object.
in	<i>src_data</i>	Source data buffer supplied to <code>init</code> .
in	<i>src_size</i>	Source buffer size supplied to <code>init</code> .
in, out	<i>dst_data</i>	Destination data buffer supplied to <code>init</code> .
in, out	<i>dst_size</i>	Destination buffer size supplied to <code>init</code> .

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

**5.12.4.11** `void te_free_operation ( te_crypto_operation_t oper )`

Frees operation internal memory.

5.12.4.12 `void te_generate_random ( void * buffer, size_t size )`

Generates random data.

5.12.4.13 `te_error_t te_get_attribute_by_id ( te_crypto_object_t object, te_attribute_id_t id, te_attribute_t ** ret )`

Finds the first attribute in the crypto object that matches ID.

#### Parameters

in	<i>object</i>	Crypto object.
in	<i>id</i>	Attribute ID to match.
out	<i>ret</i>	The attribute found.

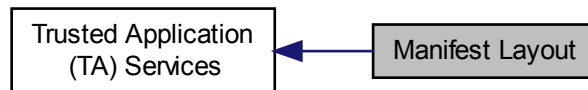
#### Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
<i>OTE_ERROR_ITEM_NOT_FOUND</i>	Indicates the requested data item was not found.

## 5.13 Manifest Layout

### 5.13.1 Detailed Description

Trusted Little Kernel (TLK) services manifest layout. Collaboration diagram for Manifest Layout:



### Data Structures

- struct [OTE\\_MANIFEST](#)

### Macros

- `#define OTE_CONFIG_MIN_STACK_SIZE(sz) OTE_CONFIG_KEY_MIN_STACK_SIZE, sz`
- `#define OTE_CONFIG_MIN_HEAP_SIZE(sz) OTE_CONFIG_KEY_MIN_HEAP_SIZE, sz`
- `#define OTE_CONFIG_MAP_MEM(id, off, sz) OTE_CONFIG_KEY_MAP_MEM, id, off, sz`
- `#define OTE_CONFIG_RESTRICT_ACCESS(clients) OTE_CONFIG_KEY_RESTRICT_ACCESS, clients`
- `#define OTE_CONFIG_AUTHORIZE(perm) OTE_CONFIG_KEY_AUTHORIZE, perm`
- `#define OTE_CONFIG_TASK_INITIAL_STATE(state) OTE_CONFIG_KEY_TASK_ISTATE, state`
- `#define OTE_MANIFEST_ATTRS __attribute__((aligned(4))) __attribute__((section(".ote.manifest")))`

### Enumerations

- enum `ote_config_key_t` {  
`OTE_CONFIG_KEY_MIN_STACK_SIZE = 1,`  
`OTE_CONFIG_KEY_MIN_HEAP_SIZE = 2,`  
`OTE_CONFIG_KEY_MAP_MEM = 3,`  
`OTE_CONFIG_KEY_RESTRICT_ACCESS = 4,`  
`OTE_CONFIG_KEY_AUTHORIZE = 5,`  
`OTE_CONFIG_KEY_TASK_ISTATE = 6` }
- enum {  
`OTE_RESTRICT_SECURE_TASKS = 1 << 0,`  
`OTE_RESTRICT_NON_SECURE_APPS = 1 << 1` }
- enum { `OTE_AUTHORIZE_INSTALL = 1 << 10` }
- enum {  
`OTE_MANIFEST_TASK_ISTATE_IMMUTABLE = 1 << 0,`  
`OTE_MANIFEST_TASK_ISTATE_STICKY = 1 << 1,`  
`OTE_MANIFEST_TASK_ISTATE_BLOCKED = 1 << 2` }

### 5.13.2 Macro Definition Documentation

#### 5.13.2.1 `#define OTE_CONFIG_MIN_STACK_SIZE( sz ) OTE_CONFIG_KEY_MIN_STACK_SIZE, sz`

Declares the minimum stack size.

Defines the minimum stack size the TLK service would expect.

#### Parameters

<i>in</i>	<i>sz</i>	The size of the stack in bytes.
-----------	-----------	---------------------------------

#### 5.13.2.2 `#define OTE_CONFIG_MIN_HEAP_SIZE( sz ) OTE_CONFIG_KEY_MIN_HEAP_SIZE, sz`

Declares the minimum heap size.

Defines the minimum heap size the TLK service would expect.

#### Parameters

<i>in</i>	<i>sz</i>	The size of the heap in bytes.
-----------	-----------	--------------------------------

#### 5.13.2.3 `#define OTE_CONFIG_MAP_MEM( id, off, sz ) OTE_CONFIG_KEY_MAP_MEM, id, off, sz`

Declares the memory address space needed.

Declares the physical memory address space the TLK service will require; a mapping will be created for TLK service.

#### Parameters

<i>in</i>	<i>id</i>	An ID number to later retrieve the mapping.
<i>in</i>	<i>off</i>	Base address of the physical address space.
<i>in</i>	<i>sz</i>	The size of the physical address space.

#### 5.13.2.4 `#define OTE_CONFIG_RESTRICT_ACCESS( clients ) OTE_CONFIG_KEY_RESTRICT_ACCESS, clients`

Declares client types that have restricted access.

#### Parameters

<i>in</i>	<i>clients</i>	A bit field to restrict access for client types.
-----------	----------------	--

#### 5.13.2.5 `#define OTE_CONFIG_AUTHORIZE( perm ) OTE_CONFIG_KEY_AUTHORIZE, perm`

Declares a privileged TA, which is allowed to load other applications and use other task administration related TLK calls.

#### Parameters

<i>in</i>	<i>perm</i>	A bit field to enable app loading.
-----------	-------------	------------------------------------

#### 5.13.2.6 `#define OTE_CONFIG_TASK_INITIAL_STATE( state ) OTE_CONFIG_KEY_TASK_ISTATE, state`

Declares that TLK must not allow the installer to override any options in the manifest.

#### Parameters

<i>in</i>	<i>state</i>	A bitfield to set the initial state to blocked.
-----------	--------------	---



5.13.2.7 `#define OTE_MANIFEST_ATTRS __attribute__((aligned(4))) __attribute__((section(\".ote.manifest\")))`

### 5.13.3 Enumeration Type Documentation

#### 5.13.3.1 `enum ote_config_key_t`

Enumerator:

***OTE\_CONFIG\_KEY\_MIN\_STACK\_SIZE***  
***OTE\_CONFIG\_KEY\_MIN\_HEAP\_SIZE***  
***OTE\_CONFIG\_KEY\_MAP\_MEM***  
***OTE\_CONFIG\_KEY\_RESTRICT\_ACCESS***  
***OTE\_CONFIG\_KEY\_AUTHORIZE***  
***OTE\_CONFIG\_KEY\_TASK\_ISTATE***

#### 5.13.3.2 anonymous enum

Defines bit flags for restricting task access by client type.

Use these bit flags with [OTE\\_CONFIG\\_RESTRICT\\_ACCESS](#) to restrict access for the type of client.

Enumerator:

***OTE\_RESTRICT\_SECURE\_TASKS***  
***OTE\_RESTRICT\_NON\_SECURE\_APPS***

#### 5.13.3.3 anonymous enum

Defines special actions that the task is authorized to perform.

To authorize a task to perform restricted operations, set the [OTE\\_CONFIG\\_AUTHORIZE](#) config bit flags to one of the following:

Currently defined authorizations are:

`OTE_AUTHORIZE_INSTALL` : Task is an installer

Enumerator:

***OTE\_AUTHORIZE\_INSTALL***

#### 5.13.3.4 anonymous enum

Defines bit flags that set attributes for the installed tasks. By default all attributes are unset.

To set a task's initial state attributes on load, set [OTE\\_CONFIG\\_KEY\\_TASK\\_ISTATE](#) config bit flags of the following-:

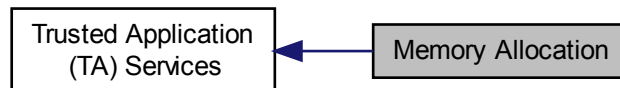
Enumerator:

***OTE\_MANIFEST\_TASK\_ISTATE\_IMMUTABLE*** Task manifest cannot be modified by the installer.  
***OTE\_MANIFEST\_TASK\_ISTATE\_STICKY*** Task cannot be unloaded.  
***OTE\_MANIFEST\_TASK\_ISTATE\_BLOCKED*** Task installed in BLOCKED state.

## 5.14 Memory Allocation

### 5.14.1 Detailed Description

Defines Trusted Little Kernel (TLK) memory allocation services functions. Collaboration diagram for Memory Allocation:



### Functions

- void \* [te\\_mem\\_alloc](#) (uint32\_t size)
- void \* [te\\_mem\\_calloc](#) (uint32\_t size)
- void [te\\_mem\\_free](#) (void \*buffer)
- void [te\\_mem\\_fill](#) (void \*buffer, uint32\_t value, uint32\_t size)
- void [te\\_mem\\_move](#) (void \*dest, void \*src, uint32\_t size)
- int [te\\_mem\\_compare](#) (void \*buffer1, void \*buffer2, uint32\_t size)

### 5.14.2 Function Documentation

#### 5.14.2.1 void\* te\_mem\_alloc ( uint32\_t size )

Allocates memory of specified size.

##### Parameters

in	size	Size in bytes of desired memory.
----	------	----------------------------------

##### Returns

A non-NULL pointer to the requested memory or NULL if the request to allocate memory failed.

#### 5.14.2.2 void\* te\_mem\_calloc ( uint32\_t size )

Allocates and clears memory of specified size.

##### Parameters

in	size	Size in bytes of desired memory.
----	------	----------------------------------

##### Returns

A non-NULL pointer to the requested memory or NULL if the request to allocate and clear memory failed.

5.14.2.3 void te\_mem\_free ( void \* *buffer* )

Frees the specified memory.

## Parameters

in	<i>buffer</i>	A pointer to memory to free.
----	---------------	------------------------------

5.14.2.4 void te\_mem\_fill ( void \* *buffer*, uint32\_t *value*, uint32\_t *size* )

Fills specified memory range with specified value.

## Parameters

in	<i>buffer</i>	A pointer to memory to fill.
in	<i>value</i>	Value to use for fill operation.
in	<i>size</i>	Number of bytes to fill.

5.14.2.5 void te\_mem\_move ( void \* *dest*, void \* *src*, uint32\_t *size* )

Moves specified number of bytes from a source memory range to a destination memory range.

## Note

The two memory ranges may overlap.

## Parameters

out	<i>dest</i>	A pointer to destination buffer for move operation.
in	<i>src</i>	A pointer to source buffer for move operation.
in	<i>size</i>	Number of bytes to move.

5.14.2.6 int te\_mem\_compare ( void \* *buffer1*, void \* *buffer2*, uint32\_t *size* )

Compares two ranges of memory for equality.

## Parameters

in	<i>buffer1</i>	A pointer to first memory range to compare.
in	<i>buffer2</i>	A pointer to second memory range to compare.
in	<i>size</i>	Number of bytes to compare.

## Returns

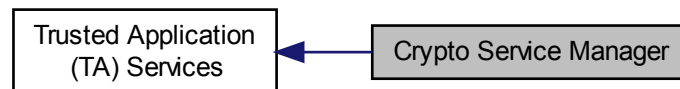
An integer where:

- 0 indicates the contents of the first memory range match the contents of the second memory range.
- < 0 indicates the contents of the first memory range are less than the contents of the second memory range.
- > 0 indicates the contents of the first memory range are greater than the contents of the second memory range.

## 5.15 Crypto Service Manager

### 5.15.1 Detailed Description

Defines APIs for managing Trusted Little Kernel (TLK) crypto services. Collaboration diagram for Crypto Service Manager:



### Functions

- [te\\_error\\_t ote\\_nvcrypto\\_init](#) (void)
- [te\\_error\\_t ote\\_nvcrypto\\_deinit](#) (void)
- [te\\_error\\_t ote\\_nvcrypto\\_get\\_keybox](#) (uint32\_t keybox\_index, void \*buf, uint32\_t \*len)
- [te\\_error\\_t ote\\_nvcrypto\\_get\\_storage\\_key](#) (uint8\_t \*key, uint32\_t key\_size)

*Gets the storage key.*

### 5.15.2 Function Documentation

#### 5.15.2.1 `te_error_t ote_nvcrypto_init ( void )`

Initializes and opens an nvcrypto service session. This function keeps track of the number of open sessions.

##### Return values

<code>OTE_SUCCESS</code>	Indicates the operation was successful.
--------------------------	---

#### 5.15.2.2 `te_error_t ote_nvcrypto_deinit ( void )`

Closes an nvcrypto service session.

##### Return values

<code>OTE_SUCCESS</code>	Indicates the operation was successful.
--------------------------	---

#### 5.15.2.3 `te_error_t ote_nvcrypto_get_keybox ( uint32_t keybox_index, void * buf, uint32_t * len )`

Gets the keybox.

##### Parameters

in	<code>keybox_index</code>	The index of the keybox.
in, out	<code>buf</code>	A pointer to the key.
in, out	<code>len</code>	The length of the buffer in bytes.

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

5.15.2.4 `te_error_t ote_nvcrypto_get_storage_key ( uint8_t * key, uint32_t key_size )`

Gets the storage key.

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

## Parameters

<i>in, out</i>	<i>key</i>	A pointer to the key.
<i>in</i>	<i>key_size</i>	The length of the key in bytes.

## 5.16 OTF Decoder Service

### 5.16.1 Detailed Description

Defines Trusted Little Kernel (TLK) on-the-fly (OTF) decoder services functions. Collaboration diagram for OTF Decoder Service:



### Functions

- [te\\_error\\_t ote\\_otf\\_init \(te\\_session\\_t \\*\\*otfSession\)](#)  
*Initializes the on-the-fly (OTF) hardware.*
- [te\\_error\\_t ote\\_otf\\_deinit \(te\\_session\\_t \\*\\*otfSession\)](#)  
*Resets the OTF hardware and erases any previous keys.*
- [te\\_error\\_t ote\\_otf\\_setkey \(void \\*buffer, uint32\\_t len, uint32\\_t \\*keySlot, te\\_session\\_t \\*otfSession\)](#)  
*Sets the key to be used by the OTF hardware.*
- [te\\_error\\_t ote\\_otf\\_erasekey \(te\\_session\\_t \\*otfSession\)](#)  
*Erases keys from the OTF hardware.*

### 5.16.2 Function Documentation

#### 5.16.2.1 [te\\_error\\_t ote\\_otf\\_init \( te\\_session\\_t \\*\\* otfSession \)](#)

Initializes the on-the-fly (OTF) hardware.

#### Parameters

<i>otfSession</i>	A pointer to the OTF session.
-------------------	-------------------------------

#### Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
<i>OTE_ERROR_BAD_STATE</i>	Indicates the session object was invalid.
<i>OTE_ERROR_OUT_OF_MEMORY</i>	Indicates the system ran out of resources.
<i>OTE_ERROR_COMMUNICATION</i>	Indicates that communication failed.

#### 5.16.2.2 [te\\_error\\_t ote\\_otf\\_deinit \( te\\_session\\_t \\*\\* otfSession \)](#)

Resets the OTF hardware and erases any previous keys.

## Parameters

<i>otfSession</i>	A pointer to the OTF session.
-------------------	-------------------------------

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
<i>OTE_ERROR_BAD_STATE</i>	Indicates the session object was invalid.

5.16.2.3 `te_error_t ote_otf_setkey ( void * buffer, uint32_t len, uint32_t * keySlot, te_session_t * otfSession )`

Sets the key to be used by the OTF hardware.

## Parameters

in	<i>buffer</i>	A pointer to the key.
in	<i>len</i>	The length of the buffer in bytes.
in, out	<i>keySlot</i>	A pointer to the key to be used.
in	<i>otfSession</i>	A pointer to the OTF session.

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
<i>OTE_ERROR_BAD_STATE</i>	Indicates the session object was invalid.
<i>OTE_ERROR_BAD_PARAMETERS</i>	Indicates input parameters were invalid.
<i>OTE_ERROR_OUT_OF_MEMORY</i>	Indicates the system ran out of resources.
<i>OTE_ERROR_COMMUNICATION</i>	Indicates that communication failed.

5.16.2.4 `te_error_t ote_otf_erasekey ( te_session_t * otfSession )`

Erases keys from the OTF hardware.

## Parameters

<i>otfSession</i>	A pointer to the OTF session.
-------------------	-------------------------------

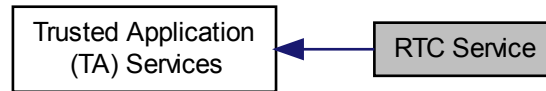
## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
<i>OTE_ERROR_BAD_STATE</i>	Indicates the session object was invalid.
<i>OTE_ERROR_OUT_OF_MEMORY</i>	Indicates the system ran out of resources.
<i>OTE_ERROR_COMMUNICATION</i>	Indicates that communication failed.

## 5.17 RTC Service

### 5.17.1 Detailed Description

Trusted Little Kernel (TLK) real-time clock (RTC) services. Collaboration diagram for RTC Service:



### Functions

- [te\\_error\\_t ote\\_rtc\\_init](#) (void)  
*Initializes the RTC hardware.*
- [te\\_error\\_t ote\\_rtc\\_deinit](#) (void)  
*Resets the RTC hardware and erases any previous keys.*
- [te\\_error\\_t ote\\_rtc\\_get\\_time](#) (uint32\_t \*rtc)  
*Gets the RTC from the RTC hardware.*

### 5.17.2 Function Documentation

#### 5.17.2.1 [te\\_error\\_t ote\\_rtc\\_init](#) ( void )

Initializes the RTC hardware.

##### Return values

<a href="#">OTE_SUCCESS</a>	Indicates the operation was successful.
-----------------------------	---

#### 5.17.2.2 [te\\_error\\_t ote\\_rtc\\_deinit](#) ( void )

Resets the RTC hardware and erases any previous keys.

##### Return values

<a href="#">OTE_SUCCESS</a>	Indicates the operation was successful.
-----------------------------	---

#### 5.17.2.3 [te\\_error\\_t ote\\_rtc\\_get\\_time](#) ( uint32\_t \* rtc )

Gets the RTC from the RTC hardware.

##### Parameters

<a href="#">out</a>	<a href="#">rtc</a>	A pointer to the RTC.
---------------------	---------------------	-----------------------



## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

## 5.18 Interface

### 5.18.1 Detailed Description

Defines Trusted Application (TA) services declarations and functions. Collaboration diagram for Interface:



### Data Structures

- struct [te\\_request\\_t](#)  
Holds the layout of the [te\\_oper\\_param\\_t](#) structures which must match the layout sent in by the non-secure (NS) world via the TrustZone Secure Monitor Call (TZ SMC) path.
- struct [te\\_ta\\_to\\_ta\\_request\\_args\\_t](#)
- struct [te\\_entry\\_point\\_message\\_t](#)
- struct [te\\_identity\\_t](#)
- struct [te\\_get\\_property\\_args\\_t](#)
- struct [te\\_device\\_unique\\_id](#)

### Macros

- `#define DEVICE_UID_SIZE_BYTES 16`
- `#define OTE_TE_FPRINTF_PREFIX_MAX_LENGTH (OTE_TASK_NAME_MAX_LENGTH + 4)`

### Enumerations

- enum {  
  [TE\\_CRITICAL](#) = 0,  
  [TE\\_ERR](#) = 1,  
  [TE\\_INFO](#) = 2,  
  [TE\\_SECURE](#) = 3,  
  [TE\\_INTERFACE](#) = 4,  
  [TE\\_RESULT](#) = 5 }
- enum {  
  [CREATE\\_INSTANCE](#) = 1UL,  
  [DESTROY\\_INSTANCE](#) = 2UL,  
  [OPEN\\_SESSION](#) = 3UL,  
  [CLOSE\\_SESSION](#) = 4UL,  
  [LAUNCH\\_OPERATION](#) = 5UL }
- enum {  
  [OTE\\_IOCTL\\_TA\\_TO\\_TA\\_REQUEST](#) = 4UL,  
  [OTE\\_IOCTL\\_GET\\_PROPERTY](#) = 5UL,  
  [OTE\\_IOCTL\\_GET\\_DEVICE\\_ID](#) = 6UL,  
  [OTE\\_IOCTL\\_GET\\_TIME\\_US](#) = 7UL,  
  [OTE\\_IOCTL\\_GET\\_RAND32](#) = 8UL,  
  [OTE\\_IOCTL\\_SS\\_REQUEST](#) = 9UL,

```
OTE_IOCTL_TASK_REQUEST = 10UL }
```

*Defines IOCTL syscall interface parameters.*

- enum {  
     TE\_PROP\_DATA\_TYPE\_UUID = 1,  
     TE\_PROP\_DATA\_TYPE\_IDENTITY = 2 }
- enum `te_property_type_t` {  
     TE\_PROPERTY\_CURRENT\_TA = 0xFFFFFFFF,  
     TE\_PROPERTY\_CURRENT\_CLIENT = 0xFFFFFFFFE,  
     TE\_PROPERTY\_TE\_IMPLEMENTATION = 0xFFFFFFFFD }

## Functions

- void `te_exit_service` (void)
- `te_error_t` `te_init` (int argc, char \*\*argv)
- void `te_destroy` (void)
- `te_error_t` `te_create_instance_iface` (void)
- void `te_destroy_instance_iface` (void)
- `te_error_t` `te_open_session_iface` (void \*\*sctx, `te_operation_t` \*oper)
- void `te_close_session_iface` (void \*sctx)
- `te_error_t` `te_receive_operation_iface` (void \*sctx, `te_operation_t` \*oper)
- void \* `ote_get_instance_data` (void)
- void `ote_set_instance_data` (void \*sessionContext)
- `te_error_t` `te_get_current_ta_uuid` (`te_service_id_t` \*value)
- `te_error_t` `te_get_client_ta_identity` (`te_identity_t` \*value)
- `te_error_t` `te_get_device_unique_id` (`te_device_unique_id` \*uid)
- void `te_fprintf_set_prefix` (const char \*prefix)
- int `te_fprintf` (int fd, char \*fmt,...) \_\_PRINTFLIKE(2)
- int void `te_oper_dump_param` (`te_oper_param_t` \*param)
- void `te_oper_dump_param_list` (`te_operation_t` \*te\_op)

## 5.18.2 Macro Definition Documentation

5.18.2.1 `#define DEVICE_UID_SIZE_BYTES 16`

5.18.2.2 `#define OTE_TE_FPRINTF_PREFIX_MAX_LENGTH (OTE_TASK_NAME_MAX_LENGTH + 4)`

Defines the maximum length of the "[task\_name] " prefix for the `te_fprintf()` task log entries.

## 5.18.3 Enumeration Type Documentation

5.18.3.1 anonymous enum

Enumerator:

**`TE_CRITICAL`**

**`TE_ERR`**

**`TE_INFO`**

**`TE_SECURE`**

**`TE_INTERFACE`**

**`TE_RESULT`**

## 5.18.3.2 anonymous enum

Enumerator:

***CREATE\_INSTANCE***  
***DESTROY\_INSTANCE***  
***OPEN\_SESSION***  
***CLOSE\_SESSION***  
***LAUNCH\_OPERATION***

## 5.18.3.3 anonymous enum

Defines IOCTL syscall interface parameters.

Enumerator:

***OTE\_IOCTL\_TA\_TO\_TA\_REQUEST***  
***OTE\_IOCTL\_GET\_PROPERTY***  
***OTE\_IOCTL\_GET\_DEVICE\_ID***  
***OTE\_IOCTL\_GET\_TIME\_US***  
***OTE\_IOCTL\_GET\_RAND32***  
***OTE\_IOCTL\_SS\_REQUEST***  
***OTE\_IOCTL\_TASK\_REQUEST***

## 5.18.3.4 anonymous enum

Defines the type of property data.

Enumerator:

***TE\_PROP\_DATA\_TYPE\_UUID***  
***TE\_PROP\_DATA\_TYPE\_IDENTITY***

## 5.18.3.5 enum te\_property\_type\_t

Defines the property data information.

Enumerator:

***TE\_PROPERTY\_CURRENT\_TA***  
***TE\_PROPERTY\_CURRENT\_CLIENT***  
***TE\_PROPERTY\_TE\_IMPLEMENTATION***

## 5.18.4 Function Documentation

## 5.18.4.1 void te\_exit\_service ( void )

## 5.18.4.2 te\_error\_t te\_init ( int argc, char \*\* argv )

Initializes the service.

## 5.18.4.3 void te\_destroy ( void )

Deinitializes the service.

## 5.18.4.4 te\_error\_t te\_create\_instance\_iface ( void )

Creates a new instance of the service.

## 5.18.4.5 void te\_destroy\_instance\_iface ( void )

Destroys an instance of the service.

## 5.18.4.6 te\_error\_t te\_open\_session\_iface ( void \*\* sctx, te\_operation\_t \* oper )

Opens a session.

## Parameters

<i>sctx</i>	A pointer to the session.
<i>oper</i>	A pointer to the operation.

## 5.18.4.7 void te\_close\_session\_iface ( void \* sctx )

Closes an opened session.

## Parameters

<i>sctx</i>	A pointer to the session to close.
-------------	------------------------------------

## 5.18.4.8 te\_error\_t te\_receive\_operation\_iface ( void \* sctx, te\_operation\_t \* oper )

Receives an operation.

## Parameters

<i>sctx</i>	A pointer to the session from which to receive the operation.
<i>oper</i>	A pointer to the operation.

## 5.18.4.9 void\* ote\_get\_instance\_data ( void )

Gets the instance context data.

## 5.18.4.10 void ote\_set\_instance\_data ( void \* sessionContext )

Sets an instance context data.

## 5.18.4.11 te\_error\_t te\_get\_current\_ta\_uuid ( te\_service\_id\_t \* value )

Gets the service ID for the current Trusted Application (TA).

## Parameters

out	value	A pointer to <a href="#">te_service_id_t</a> , which holds the service ID.
-----	-------	--

## Return values

<a href="#">OTE_SUCCESS</a>	Indicates the operation was successful.
-----------------------------	---

5.18.4.12 `te_error_t te_get_client.ta_identity ( te_identity_t * value )`

Gets the current client's identity only if it is a secure TA.

## Parameters

out	value	A pointer to <a href="#">te_identity_t</a> , which holds the client's identity.
-----	-------	---

## Return values

<a href="#">OTE_SUCCESS</a>	Indicates the operation was successful.
-----------------------------	---

5.18.4.13 `te_error_t te_get_device.unique_id ( te_device_unique_id * uid )`

Gets the device unique ID.

## Parameters

uid	A pointer to the device unique ID.
-----	------------------------------------

5.18.4.14 `void te_fprintf_set_prefix ( const char * prefix )`

Set a printable prefix string that [te\\_fprintf\(\)](#) outputs in front of every log message from this task.

The OTE library automatically sets a "[task\_name] " log prefix based on the task name set in the task manifest (if the manifest defines a task name).

## Parameters

in	prefix	The string to use for the prefix or NULL for no prefix. The maximum length of <i>prefix</i> is <a href="#">OTE_TE_FPRINTF_PREFIX_MAX_LENGTH</a> . A NULL value cancels the log prefix; a non-null string changes the prefix.
----	--------	--

5.18.4.15 `int te_fprintf ( int fd, char * fmt, ... )`

Redirects prints to Trusted Little Kernel (TLK) writes. This is a printf function for TLK services.

5.18.4.16 `int void te_oper_dump_param ( te_oper_param_t * param )`

Prints out the list of parameters for debugging.

Prints out the list of parameters with the parameter content.

## Parameters

in	param	A pointer to a TLK operation.
----	-------	-------------------------------

5.18.4.17 void `te_oper_dump_param_list` ( `te_operation_t` \* *te\_op* )

Prints out the list of parameters for debugging.

Prints out the list of parameters with the parameter content.

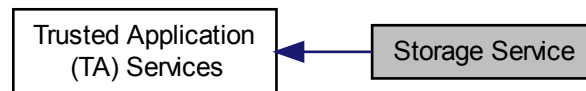
**Parameters**

<code>in</code>	<code>te_op</code>	A pointer to a TLK operation.
-----------------	--------------------	-------------------------------

## 5.19 Storage Service

### 5.19.1 Detailed Description

Defines Trusted Little Kernel (TLK) storage services declarations and functions. Collaboration diagram for Storage Service:



### Data Structures

- union [te\\_storage\\_param\\_t](#)
- struct [te\\_storage\\_request\\_t](#)

### Macros

- `#define` [TE\\_STORAGE\\_OBJID\\_MAX\\_LEN](#) 64
- `#define` [TE\\_MAX\\_STORAGE\\_REQUEST\\_PARAMS](#) 4

### Typedefs

- typedef struct  
    \_\_te\_storage\_object \* [te\\_storage\\_object\\_t](#)

### Enumerations

- enum {  
    [OTE\\_FILE\\_REQ\\_TYPE\\_CREATE](#) = 0x1,  
    [OTE\\_FILE\\_REQ\\_TYPE\\_DELETE](#) = 0x2,  
    [OTE\\_FILE\\_REQ\\_TYPE\\_OPEN](#) = 0x3,  
    [OTE\\_FILE\\_REQ\\_TYPE\\_CLOSE](#) = 0x4,  
    [OTE\\_FILE\\_REQ\\_TYPE\\_READ](#) = 0x5,  
    [OTE\\_FILE\\_REQ\\_TYPE\\_WRITE](#) = 0x6,  
    [OTE\\_FILE\\_REQ\\_TYPE\\_GET\\_SIZE](#) = 0x7,  
    [OTE\\_FILE\\_REQ\\_TYPE\\_SEEK](#) = 0x8,  
    [OTE\\_FILE\\_REQ\\_TYPE\\_TRUNC](#) = 0x9,  
    [OTE\\_FILE\\_REQ\\_TYPE\\_RPMB\\_WRITE](#) = 0x1001,  
    [OTE\\_FILE\\_REQ\\_TYPE\\_RPMB\\_READ](#) = 0x1002 }
- enum [te\\_storage\\_flags\\_t](#) {  
    [OTE\\_STORAGE\\_FLAG\\_ACCESS\\_READ](#) = 0x1,  
    [OTE\\_STORAGE\\_FLAG\\_ACCESS\\_WRITE](#) = 0x2,  
    [OTE\\_STORAGE\\_FLAG\\_ACCESS\\_WRITE\\_META](#) = 0x4 }
- enum [te\\_storage\\_whence\\_t](#) {  
    [OTE\\_STORAGE\\_SEEK\\_WHENCE\\_SET](#) = 0x1,  
    [OTE\\_STORAGE\\_SEEK\\_WHENCE\\_CUR](#) = 0x2,  
    [OTE\\_STORAGE\\_SEEK\\_WHENCE\\_END](#) = 0x3 }



## Functions

- [te\\_error\\_t te\\_create\\_storage\\_object](#) (char \*name, [te\\_storage\\_flags\\_t](#) flags, [te\\_storage\\_object\\_t](#) \*obj)
- [te\\_error\\_t te\\_open\\_storage\\_object](#) (char \*name, [te\\_storage\\_flags\\_t](#) flags, [te\\_storage\\_object\\_t](#) \*obj)
- [te\\_error\\_t te\\_read\\_storage\\_object](#) ([te\\_storage\\_object\\_t](#) obj, void \*buffer, [uint32\\_t](#) size, [uint32\\_t](#) \*count)
- [te\\_error\\_t te\\_write\\_storage\\_object](#) ([te\\_storage\\_object\\_t](#) obj, void \*buffer, [uint32\\_t](#) size)
- [te\\_error\\_t te\\_get\\_storage\\_object\\_size](#) ([te\\_storage\\_object\\_t](#) obj, [uint32\\_t](#) \*size)
- [te\\_error\\_t te\\_seek\\_storage\\_object](#) ([te\\_storage\\_object\\_t](#) obj, [int32\\_t](#) offset, [te\\_storage\\_whence\\_t](#) whence)
- [te\\_error\\_t te\\_trunc\\_storage\\_object](#) ([te\\_storage\\_object\\_t](#) obj, [uint32\\_t](#) size)
- [te\\_error\\_t te\\_delete\\_storage\\_object](#) ([te\\_storage\\_object\\_t](#) obj)
- [te\\_error\\_t te\\_close\\_storage\\_object](#) ([te\\_storage\\_object\\_t](#) obj)

## 5.19.2 Macro Definition Documentation

### 5.19.2.1 `#define TE_STORAGE_OBJID_MAX_LEN 64`

Defines the maximum file name length in bytes.

### 5.19.2.2 `#define TE_MAX_STORAGE_REQUEST_PARAMS 4`

Specifies the maximum number of parameters exchanged with the kernel storage driver.

## 5.19.3 Typedef Documentation

### 5.19.3.1 `typedef struct __te_storage_object* te_storage_object_t`

## 5.19.4 Enumeration Type Documentation

### 5.19.4.1 anonymous enum

Specifies supported file system operations.

Enumerator:

```

OTE_FILE_REQ_TYPE_CREATE
OTE_FILE_REQ_TYPE_DELETE
OTE_FILE_REQ_TYPE_OPEN
OTE_FILE_REQ_TYPE_CLOSE
OTE_FILE_REQ_TYPE_READ
OTE_FILE_REQ_TYPE_WRITE
OTE_FILE_REQ_TYPE_GET_SIZE
OTE_FILE_REQ_TYPE_SEEK
OTE_FILE_REQ_TYPE_TRUNC
OTE_FILE_REQ_TYPE_RPMB_WRITE
OTE_FILE_REQ_TYPE_RPMB_READ

```

#### 5.19.4.2 enum `te_storage_flags_t`

Defines file access flags.

Enumerator:

- `OTE_STORAGE_FLAG_ACCESS_READ`** Specifies read access.
- `OTE_STORAGE_FLAG_ACCESS_WRITE`** Specifies write access.
- `OTE_STORAGE_FLAG_ACCESS_WRITE_META`** Specifies delete access.

#### 5.19.4.3 enum `te_storage_whence_t`

Defines seek whence options.

Enumerator:

- `OTE_STORAGE_SEEK_WHENCE_SET`** Apply specified offset to start of file.
- `OTE_STORAGE_SEEK_WHENCE_CUR`** Apply specified offset to current position in file.
- `OTE_STORAGE_SEEK_WHENCE_END`** Apply specified offset to current end of file.

### 5.19.5 Function Documentation

#### 5.19.5.1 `te_error_t te_create_storage_object ( char * name, te_storage_flags_t flags, te_storage_object_t * obj )`

Creates a persistent storage object handle.

**Precondition**

This function must be called before accessing a file.

**Parameters**

in	<i>name</i>	Name of the persistent storage object (file).
in	<i>flags</i>	File access flags.
out	<i>obj</i>	A pointer to persistent object handle.

**Return values**

<b><code>OTE_SUCCESS</code></b>	Indicates the operation was successful.
---------------------------------	---

#### 5.19.5.2 `te_error_t te_open_storage_object ( char * name, te_storage_flags_t flags, te_storage_object_t * obj )`

Opens a persistent storage object.

**Precondition**

This function must be called before accessing a file.

**Parameters**

in	<i>name</i>	Name of the persistent storage object (file).
in	<i>flags</i>	File access flags.
out	<i>obj</i>	A pointer to a persistent object handle.

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

5.19.5.3 `te_error_t te_read_storage_object ( te_storage_object_t obj, void * buffer, uint32_t size, uint32_t * count )`

Reads data from a persistent object.

The actual number of bytes read can be less than the requested value and does not necessarily mean a read failure.

## Parameters

in	<i>obj</i>	Handle returned by <a href="#">te_open_storage_object()</a> .
in	<i>buffer</i>	Data buffer.
in	<i>size</i>	Size of the data buffer in bytes.
out	<i>count</i>	Actual number of bytes read.

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful and <i>count</i> contains a non-zero value.
--------------------	--

5.19.5.4 `te_error_t te_write_storage_object ( te_storage_object_t obj, void * buffer, uint32_t size )`

Writes data to the persistent object.

## Parameters

in	<i>obj</i>	Handle returned by <a href="#">te_open_storage_object()</a> .
in	<i>buffer</i>	Data buffer.
in	<i>size</i>	Size of the data buffer in bytes.

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

5.19.5.5 `te_error_t te_get_storage_object_size ( te_storage_object_t obj, uint32_t * size )`

Gets the size of the data stored in the persistent object.

## Parameters

in	<i>obj</i>	Handle returned by <a href="#">te_open_storage_object()</a> .
in	<i>size</i>	A pointer to hold the data size.

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

5.19.5.6 `te_error_t te_seek_storage_object ( te_storage_object_t obj, int32_t offset, te_storage_whence_t whence )`

Seeks to the specified offset in the persistent object.

## Parameters

in	<i>obj</i>	Handle returned by <a href="#">te_open_storage_object()</a> .
in	<i>offset</i>	Number of bytes by which to adjust the data position. A positive value specifies to adjust the data position forward while a negative value specifies to adjust it backward.
in	<i>whence</i>	The position in the data from which to calculate the new position.

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

5.19.5.7 `te_error_t te_trunc_storage_object ( te_storage_object_t obj, uint32_t size )`

Truncates the data stored in the persistent object to the specified size. If the specified size is less than the current size then any residual data are lost.

## Parameters

in	<i>obj</i>	Handle to writable persistent storage object returned by <a href="#">te_open_storage_object()</a> .
in	<i>size</i>	The new size of the object's data in bytes.

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

5.19.5.8 `te_error_t te_delete_storage_object ( te_storage_object_t obj )`

Deletes a persistent object.

## Parameters

in	<i>obj</i>	Handle returned by <a href="#">te_open_storage_object()</a> .
----	------------	---

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

5.19.5.9 `te_error_t te_close_storage_object ( te_storage_object_t obj )`

Closes the persistent object handle.

This must be called when a client is done using the handle. On completion the handle will be invalid and [te\\_open\\_storage\\_object\(\)](#) must be used again to obtain a new handle.

## Parameters

in	<i>obj</i>	Handle returned by <a href="#">te_open_storage_object()</a> .
----	------------	---

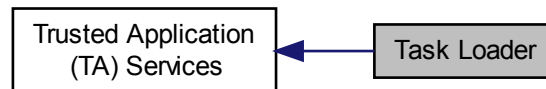
## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

## 5.20 Task Loader

### 5.20.1 Detailed Description

Defines Trusted Application (TA) services declarations and functions for task loading and management. The calls are restricted to tasks that have INSTALL permission set in the manifest. Calls from other tasks will be rejected. Collaboration diagram for Task Loader:



### Data Structures

- struct [te\\_app\\_load\\_memory\\_request\\_args\\_t](#)
- struct [te\\_task\\_info\\_t](#)
- struct [te\\_app\\_prepare\\_args\\_t](#)
- struct [te\\_task\\_restrictions\\_t](#)
- struct [te\\_app\\_start\\_args\\_t](#)
- struct [te\\_app\\_list\\_args\\_t](#)

*Holds arguments for the ioctl handler.*

- struct [te\\_list\\_apps](#)
- struct [te\\_get\\_task\\_info\\_t](#)
- struct [te\\_memory\\_mapping\\_t](#)
- struct [te\\_get\\_task\\_mapping\\_t](#)
- struct [te\\_get\\_pending\\_map\\_args\\_t](#)
- struct [te\\_system\\_info\\_args\\_t](#)
- struct [te\\_app\\_unload\\_args\\_t](#)
- struct [te\\_app\\_unload\\_t](#)
- struct [te\\_app\\_block\\_args\\_t](#)
- struct [te\\_app\\_block\\_t](#)
- struct [te\\_task\\_request\\_args\\_s](#)

### Typedefs

- typedef struct [te\\_list\\_apps](#) [te\\_list\\_apps\\_t](#)
- typedef struct [te\\_task\\_request\\_args\\_s](#) [te\\_task\\_request\\_args\\_t](#)

### Enumerations

- enum [te\\_get\\_info\\_type\\_t](#) {  
[OTE\\_GET\\_TASK\\_INFO\\_REQUEST\\_INDEX](#),  
[OTE\\_GET\\_TASK\\_INFO\\_REQUEST\\_UUID](#),  
[OTE\\_GET\\_TASK\\_INFO\\_REQUEST\\_SELF](#) }

- enum `te_app_id_t` {  
`OTE_APP_ID_INDEX`,  
`OTE_APP_ID_UUID` }
- enum `te_task_opcode_t` {  
`OTE_TASK_OP_UNKNOWN`,  
`OTE_TASK_OP_MEMORY_REQUEST`,  
`OTE_TASK_OP_PREPARE`,  
`OTE_TASK_OP_START`,  
`OTE_TASK_OP_LIST`,  
`OTE_TASK_OP_GET_TASK_INFO`,  
`OTE_TASK_OP_SYSTEM_INFO`,  
`OTE_TASK_OP_PENDING_MAPPING`,  
`OTE_TASK_OP_UNLOAD`,  
`OTE_TASK_OP_BLOCK`,  
`OTE_TASK_OP_UNBLOCK`,  
`OTE_TASK_OP_GET_MAPPING` }

## Functions

- `te_error_t te_app_request_memory` (`u_int app_size`, `uintptr_t *app_addr`, `uint32_t *app_handle`)
- `te_error_t te_app_prepare` (`uint32_t app_handle`, `te_task_info_t *app_task_info`)
- `te_error_t te_app_start` (`uint32_t app_handle`, `uint32_t app_reject`, `te_task_restrictions_t *app_restrictions`)
- `te_error_t te_list_apps` (`te_list_apps_t *app_list`)
- `te_error_t te_task_get_info` (`te_get_task_info_t *task_info`)
- `te_error_t te_task_get_mapping` (`te_get_task_mapping_t *task_mapping`)
- `te_error_t te_get_pending_task_mapping` (`uint32_t app_handle`, `te_memory_mapping_t *app_map`)
- `te_error_t te_task_get_name_self` (`char *name`, `uint32_t *len_p`)
- `te_error_t te_task_system_info` (`uint32_t type`)
- `te_error_t te_app_unload` (`te_app_unload_t *arg_unload`)
- `te_error_t te_app_block` (`te_app_block_t *app`)
- `te_error_t te_app_unblock` (`te_app_block_t *app`)

## 5.20.2 Typedef Documentation

### 5.20.2.1 typedef struct `te_list_apps` `te_list_apps_t`

Holds list command returned information.

### 5.20.2.2 typedef struct `te_task_request_args_s` `te_task_request_args_t`

Task loading ioctl argument. `IA_OPCODE` selects the operation and its argument from the union fields.

## 5.20.3 Enumeration Type Documentation

### 5.20.3.1 enum `te_get_info_type_t`

Type for the ioctl handler

Info request types:

Enumerator:

```
OTE_GET_TASK_INFO_REQUEST_INDEX
OTE_GET_TASK_INFO_REQUEST_UUID
OTE_GET_TASK_INFO_REQUEST_SELF
```

## 5.20.3.2 enum te\_app\_id\_t

Enumerator:

**OTE\_APP\_ID\_INDEX**  
**OTE\_APP\_ID\_UUID**

## 5.20.3.3 enum te\_task\_opcode\_t

Defines opcodes for OTE\_IOCTL\_TASK\_REQUEST ioctl argument for the above requests.

Enumerator:

**OTE\_TASK\_OP\_UNKNOWN**  
**OTE\_TASK\_OP\_MEMORY\_REQUEST**  
**OTE\_TASK\_OP\_PREPARE**  
**OTE\_TASK\_OP\_START**  
**OTE\_TASK\_OP\_LIST**  
**OTE\_TASK\_OP\_GET\_TASK\_INFO**  
**OTE\_TASK\_OP\_SYSTEM\_INFO**  
**OTE\_TASK\_OP\_PENDING\_MAPPING**  
**OTE\_TASK\_OP\_UNLOAD**  
**OTE\_TASK\_OP\_BLOCK**  
**OTE\_TASK\_OP\_UNBLOCK**  
**OTE\_TASK\_OP\_GET\_MAPPING**

## 5.20.4 Function Documentation

## 5.20.4.1 te\_error\_t te\_app\_request\_memory ( u\_int app\_size, uintptr\_t \* app\_addr, uint32\_t \* app\_handle )

Allocates TLK kernel memory for task loading.

Parameters

in	<i>app_size</i>	Application size in bytes.
out	<i>app_addr</i>	Contains the task virtual address of the secure memory buffer allocated by TLK (where caller can write the application image to).
out	<i>app_handle</i>	Receives the opaque handle that can be used to manage app loading. Handle not valid after app loaded or an error occurs. The handle is a 32 bit random value set by TLK.

Return values

<b>OTE_SUCCESS</b>	Indicates the operation was successful.
--------------------	---

Step 1/3 of task loading API.

## 5.20.4.2 te\_error\_t te\_app\_prepare ( uint32\_t app\_handle, te\_task\_info\_t \* app\_task\_info )

Requests the TLK to parse the static information from the task image loaded to the TLK shared memory and returns the task configuration information from the manifest section to the caller. The task is looked up by the APP\_HANDLE received from the [te\\_app\\_request\\_memory\(\)](#).

**Parameters**

in	<i>app_handle</i>	Handle for the copied application image to prepare.
out	<i>app_task_info</i>	Returns manifest data (see <a href="#">te_task_info_t</a> ) of the prepared task.

**Return values**

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

Step 2/3 of task loading API.

#### 5.20.4.3 `te_error_t te_app_start ( uint32_t app_handle, uint32_t app_reject, te_task_restrictions_t * app_restrictions )`

Starts the loaded and prepared application.

**Parameters**

in	<i>app_handle</i>	Opaque handle for the application to start After this call returns the <i>app_handle</i> is no longer valid.
in	<i>app_reject</i>	Specifies whether to reject the task. If nonzero, the task is not started but instead rejected by TLK freeing the allocated resources.
in	<i>app_restrictions</i>	Specifies overrides for a subset of task properties in manifest (unless manifest declared immutable) or NULL for no overrides. Zero field values are ignored in overrides.

**Return values**

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

**Note**

Currently this does not support "creating" a missing manifest for a task that does not have it in the binary ELF image, so also all loaded task images must contain a manifest.

Step 3/3 of task loading API.

#### 5.20.4.4 `te_error_t te_list_apps ( te_list_apps_t * app_list )`

Returns the UUID and optional name of current tasks by index.

**Parameters**

in, out	<i>app_list</i>	Indicates which task (0..N) UUID/NAME to look for.
---------	-----------------	--

**Return values**

<i>OTE_SUCCESS</i>	Indicates the operation was successful. <i>OTE_ERROR_ITEM_NOT_FOUND</i> indicates task at <i>al_index</i> is not valid.
--------------------	---

*al\_name* is a zero terminated possibly empty c-string.

#### 5.20.4.5 `te_error_t te_task_get_info ( te_get_task_info_t * task_info )`

Fetches the task config info from any task in the system.

For loaded tasks the installer gets this info also with other means, but for static tasks (like the installer itself this is



used to fetch the manifest information, particularly the `private_data` field which in the installer case configured the trust anchor (digest of root cert data blob).

Also returns current task state info in the `gti_state` and `gti_type` fields of [te\\_get\\_task\\_info\\_t](#).

#### Parameters

<code>in, out</code>	<code>task_info</code>	Request the manifest information of any task (static or loaded) <code>gti_request_type</code> and the union <code>gtiu_*</code> fields [in] identify the task and <code>gti_info</code> [out] contains the <a href="#">te_task_info_t</a> response.
----------------------	------------------------	---

#### 5.20.4.6 `te_error_t te_task_get_mapping ( te_get_task_mapping_t * task_mapping )`

Gets the specified manifest address mapping from the task.

#### Parameters

<code>in, out</code>	<code>task_mapping</code>	A pointer to manifest mapping information of any task (static or loaded). The <code>gmt_request_type</code> and the union <code>gtiu_*</code> fields [in] identify the task and <code>gmt_map</code> field <code>map_index</code> specifies the mapping (0..N).
----------------------	---------------------------	---

This function's response is in the `task_mapping` fields:

- `map_id` specifies to the mapping ID.
- `map_offset` specifies to the physical address to map.
- `map_size` specifies size of mapped area.

#### Return values

<code>OTE_SUCCESS</code>	Indicates the operation was successful.
--------------------------	---

#### 5.20.4.7 `te_error_t te_get_pending_task_mapping ( uint32_t app_handle, te_memory_mapping_t * app_map )`

Gets the specified manifest address mapping from task by its installation handle before the task is installed (this is an installer only API).

#### Parameters

<code>in</code>	<code>app_handle</code>	The handle for the pending task.
<code>in, out</code>	<code>app_map</code>	A pointer to task mapping information. <code>app_map-&gt;map_index</code> is the index [0..N] of the fetched mapping.

This function's response is in the `app_map` fields:

- `map_id` specifies to the mapping ID.
- `map_offset` specifies to the physical address to map.
- `map_size` specifies size of mapped area.

#### Return values

<code>OTE_SUCCESS</code>	Indicates the operation was successful.
--------------------------	---

#### 5.20.4.8 `te_error_t te_task_get_name_self ( char * name, uint32_t * len_p )`

Gets the current task name from its own manifest record.

##### Parameters

out	<i>name</i>	Copy task name from manifest record.
in, out	<i>len_p</i>	Contains the max length of the name on input and the length of the copied manifest name.

##### Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
--------------------	---

#### 5.20.4.9 `te_error_t te_task_system_info ( uint32_t type )`

Outputs task load system info to console, for test/debug purposes. DEBUG API only for information.

#### 5.20.4.10 `te_error_t te_app_unload ( te_app_unload_t * arg_unload )`

Unloads a previously loaded application.

##### Parameters

in	<i>arg_unload</i>	Identifies the application to unload (XX now contains UUID)
----	-------------------	---

##### Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful. <i>OTE_ERROR_ITEM_NOT_FOUND</i> indicates task with such uuid does not exist.
--------------------	---

#### 5.20.4.11 `te_error_t te_app_block ( te_app_block_t * app )`

Blocks a task.

##### Parameters

in	<i>app</i>	Identifies the application to block by INDEX or UUID.
----	------------	---

##### Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
<i>OTE_ERROR_ITEM_NOT_FOUND</i>	indicates specified task was not found. + others...

#### 5.20.4.12 `te_error_t te_app_unblock ( te_app_block_t * app )`

Unblock a previously blocked application.

##### Parameters

in	<i>app</i>	Identifies the application to unblock by INDEX or UUID
----	------------	--

## Return values

<i>OTE_SUCCESS</i>	Indicates the operation was successful.
<i>OTE_ERROR_ITEM_NOT_FOUND</i>	indicates specified task was not found. + others...



## Chapter 6

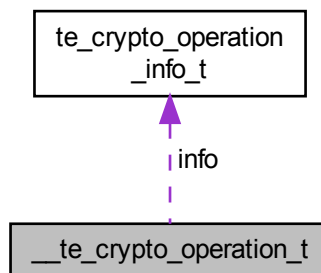
# Data Structure Documentation

### 6.1 `__te_crypto_operation_t` Struct Reference

#### 6.1.1 Detailed Description

Internal data structure for `te_crypto_operation_t`.

Collaboration diagram for `__te_crypto_operation_t`:



#### Data Fields

- [te\\_crypto\\_operation\\_info\\_t info](#)
- `void * key`
- `void * iv`
- `size_t iv_len`
- `void * imp_obj`
- [te\\_error\\_t\(\\* init\)](#)([te\\_crypto\\_operation\\_t](#) operation)
- [te\\_error\\_t\(\\* update\)](#)([te\\_crypto\\_operation\\_t](#) operation, `void *src_data`, `size_t src_size`, `void *dst_dat`, `size_t *dst_size`)
- [te\\_error\\_t\(\\* do\\_final\)](#)([te\\_crypto\\_operation\\_t](#) operation, `void *srd_data`, `size_t src_size`, `void *dst_data`, `size_t *dst_size`)
- [te\\_error\\_t\(\\* handle\\_req\)](#)([te\\_crypto\\_operation\\_t](#) operation, `void *src_data`, `size_t src_size`, `void *dst_data`, `size_t *dst_size`)
- `void(* free)`([te\\_crypto\\_operation\\_t](#) operation)

## 6.1.2 Field Documentation

6.1.2.1 `te_crypto_operation_info_t` `__te_crypto_operation_t::info`

6.1.2.2 `void*` `__te_crypto_operation_t::key`

6.1.2.3 `void*` `__te_crypto_operation_t::iv`

6.1.2.4 `size_t` `__te_crypto_operation_t::iv_len`

6.1.2.5 `void*` `__te_crypto_operation_t::imp_obj`

6.1.2.6 `te_error_t`(\* `__te_crypto_operation_t::init`)(`te_crypto_operation_t` operation)

6.1.2.7 `te_error_t`(\* `__te_crypto_operation_t::update`)(`te_crypto_operation_t` operation, `void *``src_data`, `size_t` `src_size`, `void *``dst_data`, `size_t` `dst_size`)

6.1.2.8 `te_error_t`(\* `__te_crypto_operation_t::do_final`)(`te_crypto_operation_t` operation, `void *``srd_data`, `size_t` `src_size`, `void *``dst_data`, `size_t` `dst_size`)

6.1.2.9 `te_error_t`(\* `__te_crypto_operation_t::handle_req`)(`te_crypto_operation_t` operation, `void *``src_data`, `size_t` `src_size`, `void *``dst_data`, `size_t` `dst_size`)

6.1.2.10 `void`(\* `__te_crypto_operation_t::free`)(`te_crypto_operation_t` operation)

The documentation for this struct was generated from the following file:

- [ote\\_crypto.h](#)

## 6.2 ote\_closesession Struct Reference

### 6.2.1 Detailed Description

Closes an OTE session.

#### Data Fields

- `uint32_t` [session\\_id](#)
- `cmnptr_t` [answer](#)

### 6.2.2 Field Documentation

6.2.2.1 `uint32_t` `ote_closesession::session_id`

6.2.2.2 `cmnptr_t` `ote_closesession::answer`

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.3 ote\_file\_close\_params\_t Struct Reference

## Data Fields

- `uint32_t handle`

### 6.3.1 Field Documentation

#### 6.3.1.1 `uint32_t ote_file_close_params_t::handle`

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.4 ote\_file\_create\_params\_t Struct Reference

## Data Fields

- `char dname [OTE_MAX_DIR_NAME_LEN]`
- `char fname [OTE_MAX_FILE_NAME_LEN]`
- `uint32_t flags`

### 6.4.1 Field Documentation

#### 6.4.1.1 `char ote_file_create_params_t::dname[OTE_MAX_DIR_NAME_LEN]`

#### 6.4.1.2 `char ote_file_create_params_t::fname[OTE_MAX_FILE_NAME_LEN]`

#### 6.4.1.3 `uint32_t ote_file_create_params_t::flags`

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.5 ote\_file\_delete\_params\_t Struct Reference

## Data Fields

- `char dname [OTE_MAX_DIR_NAME_LEN]`
- `char fname [OTE_MAX_FILE_NAME_LEN]`

### 6.5.1 Field Documentation

#### 6.5.1.1 `char ote_file_delete_params_t::dname[OTE_MAX_DIR_NAME_LEN]`

#### 6.5.1.2 `char ote_file_delete_params_t::fname[OTE_MAX_FILE_NAME_LEN]`

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.6 ote\_file\_get\_size\_params\_t Struct Reference

### Data Fields

- uint32\_t [handle](#)
- uint32\_t [size](#)

### 6.6.1 Field Documentation

6.6.1.1 uint32\_t ote\_file\_get\_size\_params\_t::handle

6.6.1.2 uint32\_t ote\_file\_get\_size\_params\_t::size

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.7 ote\_file\_open\_params\_t Struct Reference

### Data Fields

- char [dname](#) [OTE\_MAX\_DIR\_NAME\_LEN]
- char [fname](#) [OTE\_MAX\_FILE\_NAME\_LEN]
- uint32\_t [flags](#)
- uint32\_t [handle](#)

### 6.7.1 Field Documentation

6.7.1.1 char ote\_file\_open\_params\_t::dname[OTE\_MAX\_DIR\_NAME\_LEN]

6.7.1.2 char ote\_file\_open\_params\_t::fname[OTE\_MAX\_FILE\_NAME\_LEN]

6.7.1.3 uint32\_t ote\_file\_open\_params\_t::flags

6.7.1.4 uint32\_t ote\_file\_open\_params\_t::handle

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.8 ote\_file\_read\_params\_t Struct Reference

### Data Fields

- uint32\_t [handle](#)
- uint32\_t [data\\_size](#)
- char [data](#) [OTE\_MAX\_DATA\_SIZE]



### 6.8.1 Field Documentation

6.8.1.1 `uint32_t ote_file_read_params_t::handle`

6.8.1.2 `uint32_t ote_file_read_params_t::data_size`

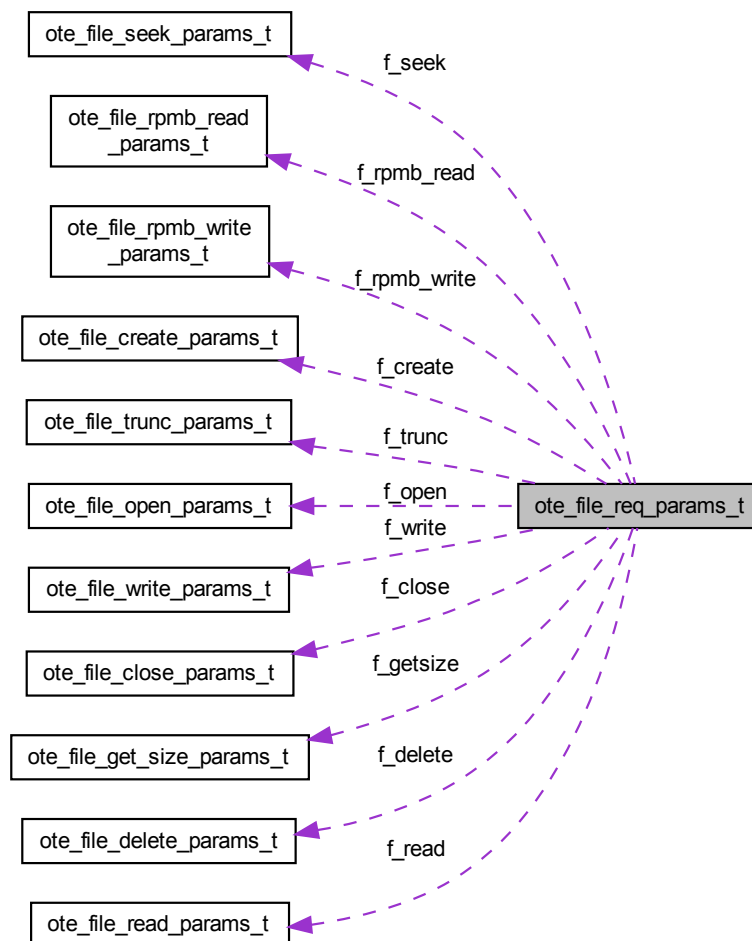
6.8.1.3 `char ote_file_read_params_t::data[OTE_MAX_DATA_SIZE]`

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.9 ote\_file\_req\_params\_t Union Reference

Collaboration diagram for `ote_file_req_params_t`:



### Data Fields

- [ote\\_file\\_create\\_params\\_t f\\_create](#)

- [ote\\_file\\_delete\\_params\\_t f\\_delete](#)
- [ote\\_file\\_open\\_params\\_t f\\_open](#)
- [ote\\_file\\_close\\_params\\_t f\\_close](#)
- [ote\\_file\\_read\\_params\\_t f\\_read](#)
- [ote\\_file\\_write\\_params\\_t f\\_write](#)
- [ote\\_file\\_seek\\_params\\_t f\\_seek](#)
- [ote\\_file\\_trunc\\_params\\_t f\\_trunc](#)
- [ote\\_file\\_get\\_size\\_params\\_t f\\_getsize](#)
- [ote\\_file\\_rpmb\\_write\\_params\\_t f\\_rpmb\\_write](#)
- [ote\\_file\\_rpmb\\_read\\_params\\_t f\\_rpmb\\_read](#)

### 6.9.1 Field Documentation

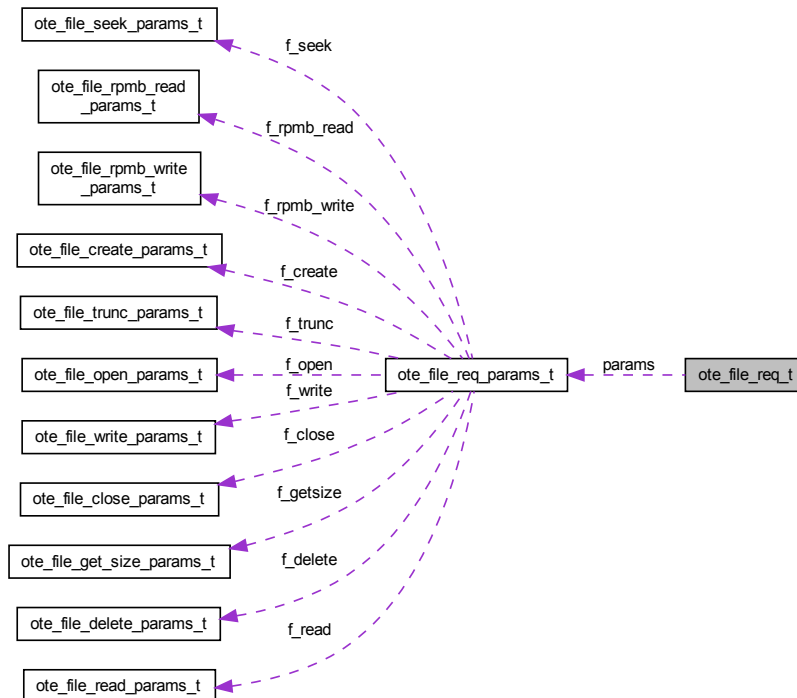
- 6.9.1.1 [ote\\_file\\_create\\_params\\_t ote\\_file\\_req\\_params.t::f\\_create](#)
- 6.9.1.2 [ote\\_file\\_delete\\_params\\_t ote\\_file\\_req\\_params.t::f\\_delete](#)
- 6.9.1.3 [ote\\_file\\_open\\_params\\_t ote\\_file\\_req\\_params.t::f\\_open](#)
- 6.9.1.4 [ote\\_file\\_close\\_params\\_t ote\\_file\\_req\\_params.t::f\\_close](#)
- 6.9.1.5 [ote\\_file\\_read\\_params\\_t ote\\_file\\_req\\_params.t::f\\_read](#)
- 6.9.1.6 [ote\\_file\\_write\\_params\\_t ote\\_file\\_req\\_params.t::f\\_write](#)
- 6.9.1.7 [ote\\_file\\_seek\\_params\\_t ote\\_file\\_req\\_params.t::f\\_seek](#)
- 6.9.1.8 [ote\\_file\\_trunc\\_params\\_t ote\\_file\\_req\\_params.t::f\\_trunc](#)
- 6.9.1.9 [ote\\_file\\_get\\_size\\_params\\_t ote\\_file\\_req\\_params.t::f\\_getsize](#)
- 6.9.1.10 [ote\\_file\\_rpmb\\_write\\_params\\_t ote\\_file\\_req\\_params.t::f\\_rpmb\\_write](#)
- 6.9.1.11 [ote\\_file\\_rpmb\\_read\\_params\\_t ote\\_file\\_req\\_params.t::f\\_rpmb\\_read](#)

The documentation for this union was generated from the following file:

- [ote\\_client.h](#)

## 6.10 ote\_file\_req\_t Struct Reference

Collaboration diagram for ote\_file\_req\_t:



### Data Fields

- `uint32_t` [type](#)
- `int32_t` [result](#)
- `uint32_t` [params\\_size](#)
- [ote\\_file\\_req\\_params\\_t](#) `params`

### 6.10.1 Field Documentation

6.10.1.1 `uint32_t` `ote_file_req_t::type`

6.10.1.2 `int32_t` `ote_file_req_t::result`

6.10.1.3 `uint32_t` `ote_file_req_t::params_size`

6.10.1.4 `ote_file_req_params_t` `ote_file_req_t::params`

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.11 ote\_file\_rpmb\_read\_params\_t Struct Reference

## Data Fields

- `uint8_t req_frame` [`OTE_RPMB_FRAME_SIZE`]
- `uint8_t resp_frame` [`OTE_RPMB_FRAME_SIZE`]

### 6.11.1 Field Documentation

6.11.1.1 `uint8_t ote_file_rpmb_read_params_t::req_frame`[`OTE_RPMB_FRAME_SIZE`]

6.11.1.2 `uint8_t ote_file_rpmb_read_params_t::resp_frame`[`OTE_RPMB_FRAME_SIZE`]

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.12 ote\_file\_rpmb\_write\_params\_t Struct Reference

### Data Fields

- `uint8_t req_frame` [`OTE_RPMB_FRAME_SIZE`]
- `uint8_t req_resp_frame` [`OTE_RPMB_FRAME_SIZE`]
- `uint8_t resp_frame` [`OTE_RPMB_FRAME_SIZE`]

### 6.12.1 Field Documentation

6.12.1.1 `uint8_t ote_file_rpmb_write_params_t::req_frame`[`OTE_RPMB_FRAME_SIZE`]

6.12.1.2 `uint8_t ote_file_rpmb_write_params_t::req_resp_frame`[`OTE_RPMB_FRAME_SIZE`]

6.12.1.3 `uint8_t ote_file_rpmb_write_params_t::resp_frame`[`OTE_RPMB_FRAME_SIZE`]

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.13 ote\_file\_seek\_params\_t Struct Reference

### Data Fields

- `uint32_t handle`
- `int32_t offset`
- `uint32_t whence`

### 6.13.1 Field Documentation

6.13.1.1 `uint32_t ote_file_seek_params_t::handle`

6.13.1.2 `int32_t ote_file_seek_params_t::offset`

6.13.1.3 `uint32_t ote_file_seek_params_t::whence`

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.14 ote\_file\_trunc\_params\_t Struct Reference

### Data Fields

- uint32\_t [handle](#)
- uint32\_t [length](#)

### 6.14.1 Field Documentation

6.14.1.1 uint32\_t ote\_file\_trunc\_params\_t::handle

6.14.1.2 uint32\_t ote\_file\_trunc\_params\_t::length

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.15 ote\_file\_write\_params\_t Struct Reference

### Data Fields

- uint32\_t [handle](#)
- uint32\_t [data\\_size](#)
- char [data](#) [[OTE\\_MAX\\_DATA\\_SIZE](#)]

### 6.15.1 Field Documentation

6.15.1.1 uint32\_t ote\_file\_write\_params\_t::handle

6.15.1.2 uint32\_t ote\_file\_write\_params\_t::data\_size

6.15.1.3 char ote\_file\_write\_params\_t::data[[OTE\\_MAX\\_DATA\\_SIZE](#)]

The documentation for this struct was generated from the following file:

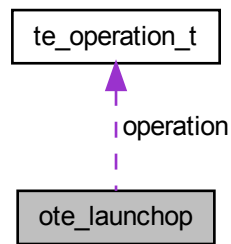
- [ote\\_client.h](#)

## 6.16 ote\_launchop Struct Reference

### 6.16.1 Detailed Description

Launches an operation request.

Collaboration diagram for `ote_launchop`:



## Data Fields

- `uint32_t session_id`
- `te_operation_t operation`
- `cmnptr_t answer`

## 6.16.2 Field Documentation

6.16.2.1 `uint32_t ote_launchop::session_id`

6.16.2.2 `te_operation_t ote_launchop::operation`

6.16.2.3 `cmnptr_t ote_launchop::answer`

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.17 OTE\_MANIFEST Struct Reference

### 6.17.1 Detailed Description

Holds the manifest structure.

The layout of the `.ote.manifest` section in the Trusted Application (TA) is the fixed fields followed by an arbitrary number of configuration options.

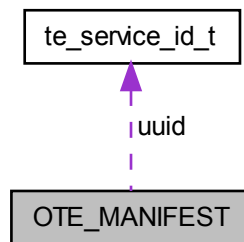
Fixed fields:

- *name* : Informative name of the task (optional).
- *private\_data* : Each TA specifies how to use this field (optional).
- *uuid* : Uniquely identifies each TA in the system.

Optional 32-bit config options:

- *config\_options[]*: Task configuration options set by macros in [ote\\_config\\_key\\_t](#).

Collaboration diagram for OTE\_MANIFEST:



## Data Fields

- char [name](#) [[OTE\\_TASK\\_NAME\\_MAX\\_LENGTH](#)]
- char [private\\_data](#) [[OTE\\_TASK\\_PRIVATE\\_DATA\\_LENGTH](#)]
- [te\\_service\\_id\\_t](#) [uuid](#)
- [uint32\\_t](#) [config\\_options](#) []

## 6.17.2 Field Documentation

6.17.2.1 char [OTE\\_MANIFEST::name](#)[[OTE\\_TASK\\_NAME\\_MAX\\_LENGTH](#)]

6.17.2.2 char [OTE\\_MANIFEST::private\\_data](#)[[OTE\\_TASK\\_PRIVATE\\_DATA\\_LENGTH](#)]

6.17.2.3 [te\\_service\\_id\\_t](#) [OTE\\_MANIFEST::uuid](#)

6.17.2.4 [uint32\\_t](#) [OTE\\_MANIFEST::config\\_options](#) []

The documentation for this struct was generated from the following file:

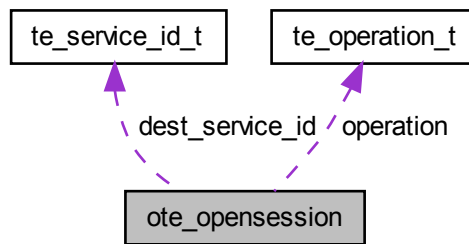
- [ote\\_manifest.h](#)

## 6.18 ote\_opensession Struct Reference

### 6.18.1 Detailed Description

Opens an open trusted environment (OTE) session.

Collaboration diagram for ote\_opensession:



## Data Fields

- [te\\_service\\_id\\_t](#) `dest_service_id`
- [te\\_operation\\_t](#) `operation`
- [cmnptr\\_t](#) `answer`

## 6.18.2 Field Documentation

6.18.2.1 [te\\_service\\_id\\_t](#) `ote_opensession::dest_service_id`

6.18.2.2 [te\\_operation\\_t](#) `ote_opensession::operation`

6.18.2.3 [cmnptr\\_t](#) `ote_opensession::answer`

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.19 ote\_ss\_op\_t Struct Reference

### Data Fields

- [uint32\\_t](#) `req_size`
- [uint8\\_t](#) `data` [[SS\\_OP\\_MAX\\_DATA\\_SIZE](#)]

### 6.19.1 Field Documentation

6.19.1.1 [uint32\\_t](#) `ote_ss_op_t::req_size`

6.19.1.2 [uint8\\_t](#) `ote_ss_op_t::data`[[SS\\_OP\\_MAX\\_DATA\\_SIZE](#)]

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)



## 6.20 te\_answer Struct Reference

### Data Fields

- uint32\_t [result](#)
- uint32\_t [session\\_id](#)
- uint32\_t [result\\_origin](#)

### 6.20.1 Field Documentation

6.20.1.1 uint32\_t te\_answer::result

6.20.1.2 uint32\_t te\_answer::session\_id

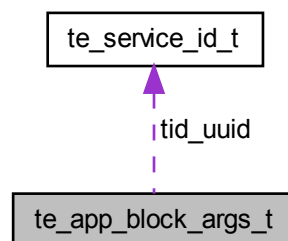
6.20.1.3 uint32\_t te\_answer::result\_origin

The documentation for this struct was generated from the following file:

- [ote\\_client.h](#)

## 6.21 te\_app\_block\_args\_t Struct Reference

Collaboration diagram for te\_app\_block\_args\_t:



### Data Fields

- [te\\_app\\_id\\_t](#) tid\_type
- union {
  - uint32\_t [tid\\_index](#)
  - [te\\_service\\_id\\_t](#) tid\_uuid
- };

### 6.21.1 Field Documentation

6.21.1.1 [te\\_app\\_id\\_t](#) te\_app\_block\_args\_t::tid\_type

6.21.1.2 `uint32_t te_app_block_args_t::tid_index`

6.21.1.3 `te_service_id_t te_app_block_args_t::tid_uuid`

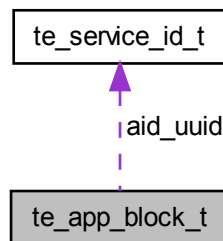
6.21.1.4 `union { ... }`

The documentation for this struct was generated from the following file:

- [ote\\_task\\_load.h](#)

## 6.22 `te_app_block_t` Struct Reference

Collaboration diagram for `te_app_block_t`:



### Data Fields

- [te\\_app\\_id\\_t aid\\_type](#)
- `union {`  
     [uint32\\_t aid\\_index](#)  
     [te\\_service\\_id\\_t aid\\_uuid](#)  
   };

### 6.22.1 Field Documentation

6.22.1.1 `te_app_id_t te_app_block_t::aid_type`

6.22.1.2 `uint32_t te_app_block_t::aid_index`

6.22.1.3 `te_service_id_t te_app_block_t::aid_uuid`

6.22.1.4 `union { ... }`

The documentation for this struct was generated from the following file:

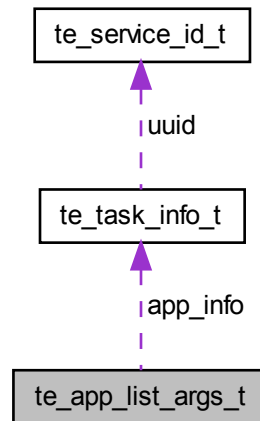
- [ote\\_task\\_load.h](#)

## 6.23 `te_app_list_args_t` Struct Reference

### 6.23.1 Detailed Description

Holds arguments for the ioctl handler.

Collaboration diagram for `te_app_list_args_t`:



### Data Fields

- `uint32_t` [app\\_index](#)
- `uint32_t` [app\\_type](#)
- `uint32_t` [app\\_state](#)
- [te\\_task\\_info\\_t](#) `app_info`

### 6.23.2 Field Documentation

6.23.2.1 `uint32_t` `te_app_list_args_t::app_index`

6.23.2.2 `uint32_t` `te_app_list_args_t::app_type`

6.23.2.3 `uint32_t` `te_app_list_args_t::app_state`

6.23.2.4 `te_task_info_t` `te_app_list_args_t::app_info`

The documentation for this struct was generated from the following file:

- [ote\\_task\\_load.h](#)

## 6.24 `te_app_load_memory_request_args_t` Struct Reference

### 6.24.1 Detailed Description

Holds the ioctl handler.

#### Data Fields

- uint32\_t [app\\_handle](#)
- uintptr\_t [app\\_addr](#)
- uint32\_t [app\\_size](#)

### 6.24.2 Field Documentation

6.24.2.1 uint32\_t te\_app\_load\_memory\_request\_args\_t::app\_handle

6.24.2.2 uintptr\_t te\_app\_load\_memory\_request\_args\_t::app\_addr

6.24.2.3 uint32\_t te\_app\_load\_memory\_request\_args\_t::app\_size

The documentation for this struct was generated from the following file:

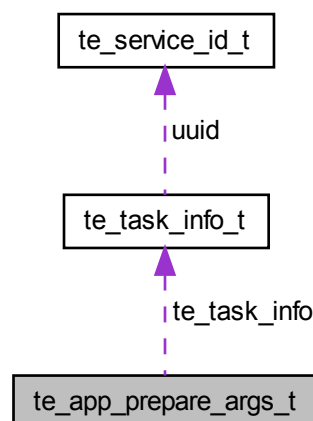
- [ote\\_task\\_load.h](#)

## 6.25 te\_app\_prepare\_args\_t Struct Reference

### 6.25.1 Detailed Description

type for the ioctl handler

Collaboration diagram for te\_app\_prepare\_args\_t:



#### Data Fields

- uint32\_t [app\\_handle](#)

- [te\\_task\\_info\\_t te\\_task\\_info](#)

### 6.25.2 Field Documentation

6.25.2.1 `uint32_t te_app_prepare_args_t::app_handle`

6.25.2.2 `te_task_info_t te_app_prepare_args_t::te_task_info`

The documentation for this struct was generated from the following file:

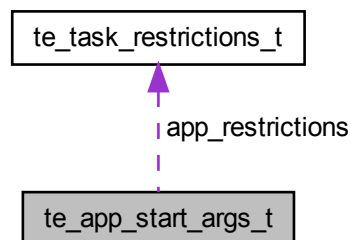
- [ote\\_task\\_load.h](#)

## 6.26 te\_app\_start\_args\_t Struct Reference

### 6.26.1 Detailed Description

type for the ioctl handler

Collaboration diagram for `te_app_start_args_t`:



### Data Fields

- `uint32_t app_handle`
- `uint32_t app_reject`
- `te_task_restrictions_t app_restrictions`

### 6.26.2 Field Documentation

6.26.2.1 `uint32_t te_app_start_args_t::app_handle`

6.26.2.2 `uint32_t te_app_start_args_t::app_reject`

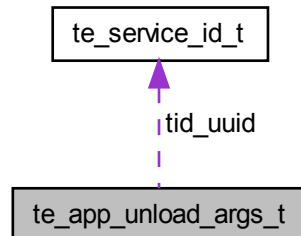
6.26.2.3 `te_task_restrictions_t te_app_start_args_t::app_restrictions`

The documentation for this struct was generated from the following file:

- [ote\\_task\\_load.h](#)

## 6.27 te\_app\_unload\_args\_t Struct Reference

Collaboration diagram for `te_app_unload_args_t`:



### Data Fields

- [te\\_app\\_id\\_t](#) `tid_type`
- union {
  - [uint32\\_t](#) `tid_index`
  - [te\\_service\\_id\\_t](#) `tid_uuid`

### 6.27.1 Field Documentation

6.27.1.1 `te_app_id_t` `te_app_unload_args_t::tid_type`

6.27.1.2 `uint32_t` `te_app_unload_args_t::tid_index`

6.27.1.3 `te_service_id_t` `te_app_unload_args_t::tid_uuid`

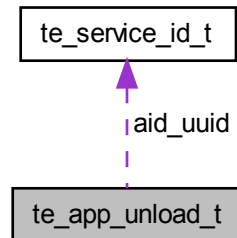
6.27.1.4 union { ... }

The documentation for this struct was generated from the following file:

- [ote\\_task\\_load.h](#)

## 6.28 te\_app\_unload\_t Struct Reference

Collaboration diagram for te\_app\_unload\_t:



### Data Fields

- [te\\_app\\_id\\_t](#) aid\_type
- union {
  - uint32\_t [aid\\_index](#)
  - [te\\_service\\_id\\_t](#) aid\_uuid

### 6.28.1 Field Documentation

6.28.1.1 [te\\_app\\_id\\_t](#) te\_app\_unload\_t::aid\_type

6.28.1.2 uint32\_t te\_app\_unload\_t::aid\_index

6.28.1.3 [te\\_service\\_id\\_t](#) te\_app\_unload\_t::aid\_uuid

6.28.1.4 union { ... }

The documentation for this struct was generated from the following file:

- [ote\\_task\\_load.h](#)

## 6.29 te\_attribute\_t Struct Reference

### 6.29.1 Detailed Description

Defines attribute object internals. Attributes can be integer or reference type.

### Data Fields

- [te\\_attribute\\_id\\_t](#) id

```

• union {
    struct {
        void * buffer
        size_t size
    } ref
    struct {
        uint32_t a
        uint32_t b
    } value
} content

```

## 6.29.2 Field Documentation

6.29.2.1 [te\\_attribute\\_id\\_t](#) [te\\_attribute\\_t::id](#)

6.29.2.2 [void\\*](#) [te\\_attribute\\_t::buffer](#)

6.29.2.3 [size\\_t](#) [te\\_attribute\\_t::size](#)

6.29.2.4 [struct { ... }](#) [te\\_attribute\\_t::ref](#)

6.29.2.5 [uint32\\_t](#) [te\\_attribute\\_t::a](#)

6.29.2.6 [uint32\\_t](#) [te\\_attribute\\_t::b](#)

6.29.2.7 [struct { ... }](#) [te\\_attribute\\_t::value](#)

6.29.2.8 [union { ... }](#) [te\\_attribute\\_t::content](#)

The documentation for this struct was generated from the following file:

- [ote\\_attrs.h](#)

## 6.30 [te\\_cache\\_maint\\_args\\_t](#) Struct Reference

### 6.30.1 Detailed Description

Holds an op code and data used to for cache maintenance.

Used with the [OTE\\_IOCTL\\_CACHE\\_MAINT](#) operation.

### Data Fields

- [void \\*](#) [vaddr](#)  
*Holds a pointer to the virtual address.*
- [uint32\\_t](#) [length](#)  
*Holds the length of the address space.*
- [uint32\\_t](#) [op](#)  
*Holds the operation; see [te\\_ext\\_nv\\_cache\\_maint\\_op\\_t](#).*



### 6.30.2 Field Documentation

#### 6.30.2.1 void\* te\_cache\_maint\_args\_t::vaddr

Holds a pointer to the virtual address.

#### 6.30.2.2 uint32\_t te\_cache\_maint\_args\_t::length

Holds the length of the address space.

#### 6.30.2.3 uint32\_t te\_cache\_maint\_args\_t::op

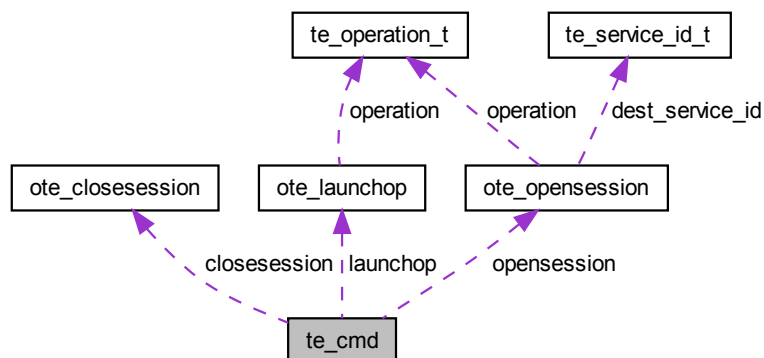
Holds the operation; see [te\\_ext\\_nv\\_cache\\_maint\\_op\\_t](#).

The documentation for this struct was generated from the following file:

- [ote\\_ext\\_nv.h](#)

## 6.31 te\_cmd Union Reference

Collaboration diagram for te\_cmd:



### Data Fields

- struct [ote\\_opensession](#) `opensession`
- struct [ote\\_closesession](#) `closesession`
- struct [ote\\_launchop](#) `launchop`

### 6.31.1 Field Documentation

#### 6.31.1.1 struct ote\_opensession te\_cmd::opensession

#### 6.31.1.2 struct ote\_closesession te\_cmd::closesession

### 6.31.1.3 struct `ote_launchop_t`:`launchop`

The documentation for this union was generated from the following file:

- [ote\\_client.h](#)

## 6.32 te\_crypto\_operation\_info\_t Struct Reference

### 6.32.1 Detailed Description

Holds a crypto operation info object.

#### Data Fields

- `uint32_t` [algorithm](#)
- `uint32_t` [operation\\_class](#)
- `uint32_t` [mode](#)
- `uint32_t` [digest\\_length](#)
- `uint32_t` [key\\_size](#)
- `uint32_t` [required\\_key\\_usage](#)
- `uint32_t` [handle\\_state](#)

### 6.32.2 Field Documentation

6.32.2.1 `uint32_t` `te_crypto_operation_info_t::algorithm`

6.32.2.2 `uint32_t` `te_crypto_operation_info_t::operation_class`

6.32.2.3 `uint32_t` `te_crypto_operation_info_t::mode`

6.32.2.4 `uint32_t` `te_crypto_operation_info_t::digest_length`

6.32.2.5 `uint32_t` `te_crypto_operation_info_t::key_size`

6.32.2.6 `uint32_t` `te_crypto_operation_info_t::required_key_usage`

6.32.2.7 `uint32_t` `te_crypto_operation_info_t::handle_state`

The documentation for this struct was generated from the following file:

- [ote\\_crypto.h](#)

## 6.33 te\_crypto\_rsa\_key\_t Struct Reference

### 6.33.1 Detailed Description

Holds internal data for RSA keys.

## Data Fields

- `uint8_t * public_mod`
- `int public_mod_len`
- `uint8_t * public_expo`
- `int public_expo_len`
- `uint8_t * private_expo`
- `int private_expo_len`
- `uint8_t * prime1`
- `int prime1_len`
- `uint8_t * prime2`
- `int prime2_len`
- `uint8_t * expo1`
- `int expo1_len`
- `uint8_t * expo2`
- `int expo2_len`
- `uint8_t * coeff`
- `int coeff_len`

### 6.33.2 Field Documentation

#### 6.33.2.1 `uint8_t* te_crypto_rsa_key_t::public_mod`

Holds public modulus.

#### 6.33.2.2 `int te_crypto_rsa_key_t::public_mod_len`

Holds public modulus length in bytes.

#### 6.33.2.3 `uint8_t* te_crypto_rsa_key_t::public_expo`

Holds public exponent.

#### 6.33.2.4 `int te_crypto_rsa_key_t::public_expo_len`

Holds public exponent length in bytes.

#### 6.33.2.5 `uint8_t* te_crypto_rsa_key_t::private_expo`

Holds private exponent.

#### 6.33.2.6 `int te_crypto_rsa_key_t::private_expo_len`

Holds private exponent length in bytes.

#### 6.33.2.7 `uint8_t* te_crypto_rsa_key_t::prime1`

Holds secret prime factor.

#### 6.33.2.8 `int te_crypto_rsa_key_t::prime1_len`

Holds *prime1* length in bytes.

#### 6.33.2.9 `uint8_t* te_crypto_rsa_key_t::prime2`

Holds secret prime factor.

#### 6.33.2.10 `int te_crypto_rsa_key_t::prime2_len`

Holds *prime2* length in bytes.

#### 6.33.2.11 `uint8_t* te_crypto_rsa_key_t::expo1`

Holds  $d \bmod (p-1)$ .

#### 6.33.2.12 `int te_crypto_rsa_key_t::expo1_len`

Holds *expo1* length in bytes.

#### 6.33.2.13 `uint8_t* te_crypto_rsa_key_t::expo2`

Holds  $d \bmod (q-1)$ .

#### 6.33.2.14 `int te_crypto_rsa_key_t::expo2_len`

Holds *expo2* length in bytes.

#### 6.33.2.15 `uint8_t* te_crypto_rsa_key_t::coeff`

Holds  $q^{-1} \bmod p$ .

#### 6.33.2.16 `int te_crypto_rsa_key_t::coeff_len`

Holds the coefficient length in bytes.

The documentation for this struct was generated from the following file:

- [ote\\_crypto.h](#)

## 6.34 `te_device_unique_id` Struct Reference

### 6.34.1 Detailed Description

Holds the device unique ID.

#### Data Fields

- `uint8_t id` [[DEVICE\\_UID\\_SIZE\\_BYTES](#)]

### 6.34.2 Field Documentation

#### 6.34.2.1 uint8\_t te\_device\_unique\_id::id[DEVICE\_UID\_SIZE\_BYTES]

The documentation for this struct was generated from the following file:

- [ote\\_service.h](#)

## 6.35 te\_entry\_point\_message\_t Struct Reference

### Data Fields

- uint32\_t [type](#)
- [te\\_error\\_t](#) [result](#)
- [cmnptr\\_t](#) [context](#)
- uint32\_t [command\\_id](#)
- [cmnptr\\_t](#) [params](#)
- uint32\_t [params\\_size](#)

### 6.35.1 Field Documentation

#### 6.35.1.1 uint32\_t te\_entry\_point\_message\_t::type

#### 6.35.1.2 [te\\_error\\_t](#) te\_entry\_point\_message\_t::result

#### 6.35.1.3 [cmnptr\\_t](#) te\_entry\_point\_message\_t::context

#### 6.35.1.4 uint32\_t te\_entry\_point\_message\_t::command\_id

#### 6.35.1.5 [cmnptr\\_t](#) te\_entry\_point\_message\_t::params

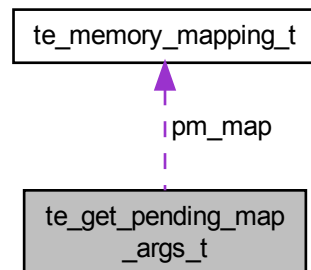
#### 6.35.1.6 uint32\_t te\_entry\_point\_message\_t::params\_size

The documentation for this struct was generated from the following file:

- [ote\\_service.h](#)

## 6.36 `te_get_pending_map_args_t` Struct Reference

Collaboration diagram for `te_get_pending_map_args_t`:



### Data Fields

- `uint32_t` [pm\\_handle](#)
- [te\\_memory\\_mapping\\_t](#) `pm_map`

### 6.36.1 Field Documentation

6.36.1.1 `uint32_t` `te_get_pending_map_args_t::pm_handle`

6.36.1.2 `te_memory_mapping_t` `te_get_pending_map_args_t::pm_map`

The documentation for this struct was generated from the following file:

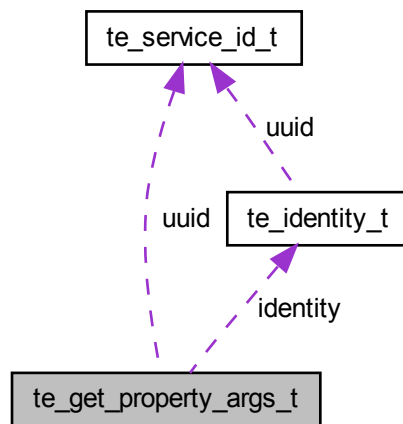
- [ote\\_task\\_load.h](#)

## 6.37 `te_get_property_args_t` Struct Reference

### 6.37.1 Detailed Description

Holds data about the TA client.

Collaboration diagram for te\_get\_property\_args\_t:



## Data Fields

- [te\\_property\\_type\\_t](#) `prop`  
Holds the `TE_PROPERTY_*` value.
- `uint32_t` [data\\_type](#)  
Holds the data type of property.
- union {  
    [te\\_service\\_id\\_t](#) `uuid`  
    [te\\_identity\\_t](#) `identity`  
} `value`  
  
Holds the return value.
- `size_t` [value\\_size](#)  
Holds the size of return value.
- [te\\_error\\_t](#) `result`

## 6.37.2 Field Documentation

### 6.37.2.1 `te_property_type_t te_get_property_args_t::prop`

Holds the `TE_PROPERTY_*` value.

### 6.37.2.2 `uint32_t te_get_property_args_t::data_type`

Holds the data type of property.

### 6.37.2.3 `te_service_id_t te_get_property_args_t::uuid`

### 6.37.2.4 `te_identity_t te_get_property_args_t::identity`

#### 6.37.2.5 `union { ... } te_get_property_args_t::value`

Holds the return value.

#### 6.37.2.6 `size_t te_get_property_args_t::value_size`

Holds the size of return value.

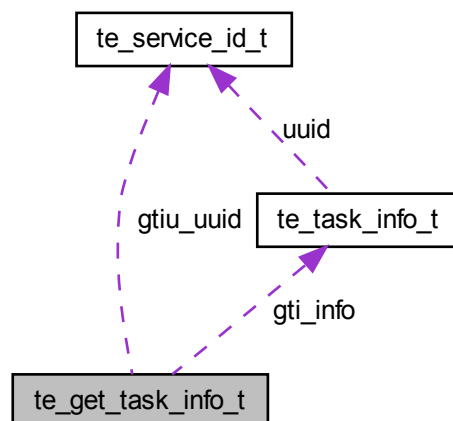
#### 6.37.2.7 `te_error_t te_get_property_args_t::result`

The documentation for this struct was generated from the following file:

- [ote\\_service.h](#)

## 6.38 `te_get_task_info_t` Struct Reference

Collaboration diagram for `te_get_task_info_t`:



### Data Fields

- [te\\_get\\_info\\_type\\_t gti\\_request\\_type](#)
- `union {`  
     [uint32\\_t gtiu\\_index](#)  
     [te\\_service\\_id\\_t gtiu\\_uuid](#)  
   };
- [uint32\\_t gti\\_state](#)
- [uint32\\_t gti\\_type](#)
- [te\\_task\\_info\\_t gti\\_info](#)



### 6.38.1 Field Documentation

6.38.1.1 `te_get_info_type_t te_get_task_info_t::gti_request_type`

6.38.1.2 `uint32_t te_get_task_info_t::gtiu_index`

6.38.1.3 `te_service_id_t te_get_task_info_t::gtiu_uuid`

6.38.1.4 `union { ... }`

6.38.1.5 `uint32_t te_get_task_info_t::gti_state`

6.38.1.6 `uint32_t te_get_task_info_t::gti_type`

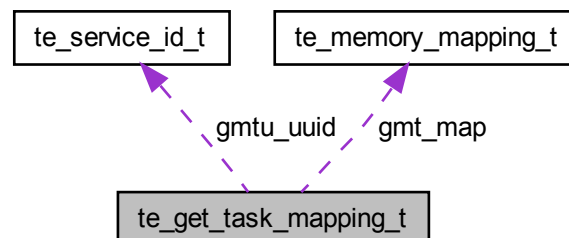
6.38.1.7 `te_task_info_t te_get_task_info_t::gti_info`

The documentation for this struct was generated from the following file:

- [ote\\_task\\_load.h](#)

## 6.39 te\_get\_task\_mapping\_t Struct Reference

Collaboration diagram for `te_get_task_mapping_t`:



### Data Fields

- [te\\_get\\_info\\_type\\_t gmt\\_request\\_type](#)
- `union {`  
     [uint32\\_t gmtu\\_index](#)  
     [te\\_service\\_id\\_t gmtu\\_uuid](#)  
   };
- [te\\_memory\\_mapping\\_t gmt\\_map](#)

### 6.39.1 Field Documentation

6.39.1.1 `te_get_info_type_t te_get_task_mapping_t::gmt_request_type`

6.39.1.2 `uint32_t te_get_task_mapping_t::gmtu_index`

6.39.1.3 `te_service_id_t te_get_task_mapping_t::gmtu_uuid`

6.39.1.4 `union { ... }`

6.39.1.5 `te_memory_mapping_t te_get_task_mapping_t::gmt_map`

The documentation for this struct was generated from the following file:

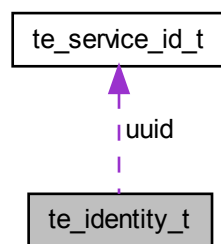
- [ote\\_task\\_load.h](#)

## 6.40 `te_identity_t` Struct Reference

### 6.40.1 Detailed Description

Holds the identity of a client/caller.

Collaboration diagram for `te_identity_t`:



### Data Fields

- `uint32_t login`
- `te_service_id_t uuid`

### 6.40.2 Field Documentation

6.40.2.1 `uint32_t te_identity_t::login`

6.40.2.2 `te_service_id_t te_identity_t::uuid`

The documentation for this struct was generated from the following file:

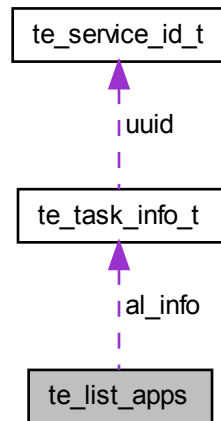
- [ote\\_service.h](#)

## 6.41 `te_list_apps` Struct Reference

### 6.41.1 Detailed Description

Holds list command returned information.

Collaboration diagram for te\_list\_apps:



### Data Fields

- `uint32_t al_index`
- `uint32_t al_type`
- `uint32_t al_state`
- `te_task_info_t al_info`

### 6.41.2 Field Documentation

6.41.2.1 `uint32_t te_list_apps::al_index`

6.41.2.2 `uint32_t te_list_apps::al_type`

6.41.2.3 `uint32_t te_list_apps::al_state`

6.41.2.4 `te_task_info_t te_list_apps::al_info`

The documentation for this struct was generated from the following file:

- [ote\\_task\\_load.h](#)

## 6.42 te\_map\_mem\_addr\_args\_t Struct Reference

### 6.42.1 Detailed Description

Holds a pointer to the map memory for a specific `OTE_CONFIG_MAP_MEM` ID value.

Used with the `OTE_IOCTL_GET_MAP_MEM_ADDR` operation.

## Data Fields

- `uint32_t id`  
*Holds the [OTE\\_CONFIG\\_MAP\\_MEM](#) ID value.*
- `void * addr`  
*Holds a pointer to the corresponding address of mapping.*

### 6.42.2 Field Documentation

#### 6.42.2.1 `uint32_t te_map_mem_addr_args_t::id`

Holds the [OTE\\_CONFIG\\_MAP\\_MEM](#) ID value.

#### 6.42.2.2 `void* te_map_mem_addr_args_t::addr`

Holds a pointer to the corresponding address of mapping.

The documentation for this struct was generated from the following file:

- [ote\\_ext\\_nv.h](#)

## 6.43 `te_memory_mapping_t` Struct Reference

### Data Fields

- `uint32_t map_index`
- `uint32_t map_id`
- `uint32_t map_offset`
- `uint32_t map_size`

### 6.43.1 Field Documentation

#### 6.43.1.1 `uint32_t te_memory_mapping_t::map_index`

#### 6.43.1.2 `uint32_t te_memory_mapping_t::map_id`

#### 6.43.1.3 `uint32_t te_memory_mapping_t::map_offset`

#### 6.43.1.4 `uint32_t te_memory_mapping_t::map_size`

The documentation for this struct was generated from the following file:

- [ote\\_task\\_load.h](#)

## 6.44 `te_oper_param_t` Struct Reference

### 6.44.1 Detailed Description

Holds the operation object parameters.

## Data Fields

- uint32\_t [index](#)
- [te\\_oper\\_param\\_type\\_t](#) type
- union {
  - struct {
    - uint32\_t [val](#)
  - Int
  - struct {
    - [cmnptr\\_t](#) base
    - uint32\_t [len](#)
  - Mem
- u
- [cmnptr\\_t](#) next

### 6.44.2 Field Documentation

6.44.2.1 uint32\_t te\_oper\_param\_t::index

6.44.2.2 [te\\_oper\\_param\\_type\\_t](#) te\_oper\_param\_t::type

6.44.2.3 uint32\_t te\_oper\_param\_t::val

6.44.2.4 struct { ... } te\_oper\_param\_t::Int

6.44.2.5 [cmnptr\\_t](#) te\_oper\_param\_t::base

6.44.2.6 uint32\_t te\_oper\_param\_t::len

6.44.2.7 struct { ... } te\_oper\_param\_t::Mem

6.44.2.8 union { ... } te\_oper\_param\_t::u

6.44.2.9 [cmnptr\\_t](#) te\_oper\_param\_t::next

The documentation for this struct was generated from the following file:

- [ote\\_common.h](#)

## 6.45 te\_operation\_t Struct Reference

### 6.45.1 Detailed Description

Holds operation object information that is to be delivered to the TLK Secure Service.

## Data Fields

- uint32\_t [command](#)
- [te\\_error\\_t](#) status
- [cmnptr\\_t](#) list\_head
  - Holds pointers to the head/tail of the list of param\_t nodes.*
- [cmnptr\\_t](#) list\_tail

- uint32\_t [list\\_count](#)
- uint32\_t [interface\\_side](#)

## 6.45.2 Field Documentation

6.45.2.1 uint32\_t te\_operation\_t::command

6.45.2.2 te\_error\_t te\_operation\_t::status

6.45.2.3 cmnptr\_t te\_operation\_t::list\_head

Holds pointers to the head/tail of the list of *param\_t* nodes.

6.45.2.4 cmnptr\_t te\_operation\_t::list\_tail

6.45.2.5 uint32\_t te\_operation\_t::list\_count

6.45.2.6 uint32\_t te\_operation\_t::interface\_side

The documentation for this struct was generated from the following file:

- [ote\\_common.h](#)

## 6.46 te\_request\_t Struct Reference

### 6.46.1 Detailed Description

Holds the layout of the [te\\_oper\\_param\\_t](#) structures which must match the layout sent in by the non-secure (NS) world via the TrustZone Secure Monitor Call (TZ SMC) path.

### Data Fields

- uint32\_t [type](#)
- uint32\_t [session\\_id](#)
- uint32\_t [command\\_id](#)
- cmnptr\_t [params](#)
- uint32\_t [params\\_size](#)
- uint32\_t [dest\\_uuid](#) [4]
- uint32\_t [result](#)
- uint32\_t [result\\_origin](#)

## 6.46.2 Field Documentation

6.46.2.1 uint32\_t te\_request\_t::type

6.46.2.2 uint32\_t te\_request\_t::session\_id

6.46.2.3 uint32\_t te\_request\_t::command\_id

6.46.2.4 cmnptr\_t te\_request\_t::params

6.46.2.5 `uint32_t te_request_t::params_size`

6.46.2.6 `uint32_t te_request_t::dest_uuid[4]`

6.46.2.7 `uint32_t te_request_t::result`

6.46.2.8 `uint32_t te_request_t::result_origin`

The documentation for this struct was generated from the following file:

- [ote\\_service.h](#)

## 6.47 `te_service_id_t` Struct Reference

### 6.47.1 Detailed Description

Defines a unique 16-byte ID for each TLK service.

#### Data Fields

- `uint32_t time_low`
- `uint16_t time_mid`
- `uint16_t time_hi_and_version`
- `uint8_t clock_seq_and_node [8]`

### 6.47.2 Field Documentation

6.47.2.1 `uint32_t te_service_id_t::time_low`

6.47.2.2 `uint16_t te_service_id_t::time_mid`

6.47.2.3 `uint16_t te_service_id_t::time_hi_and_version`

6.47.2.4 `uint8_t te_service_id_t::clock_seq_and_node[8]`

The documentation for this struct was generated from the following file:

- [ote\\_common.h](#)

## 6.48 `te_session_t` Union Reference

### 6.48.1 Detailed Description

Holds session information.

#### Data Fields

- struct {  
    `uint32_t session_id`  
    `uint32_t context_id`  
    `te_result_origin_t result_origin`

```

    } client

    • struct {
        uint32_t session_id
        te_result_origin_t result_origin
    } service

```

## 6.48.2 Field Documentation

6.48.2.1 `uint32_t te_session_t::session_id`

6.48.2.2 `uint32_t te_session_t::context_id`

6.48.2.3 `te_result_origin_t te_session_t::result_origin`

6.48.2.4 `struct { ... } te_session_t::client`

6.48.2.5 `struct { ... } te_session_t::service`

The documentation for this union was generated from the following file:

- [ote\\_common.h](#)

## 6.49 `te_storage_param_t` Union Reference

### 6.49.1 Detailed Description

Holds a storage operation parameter used in [te\\_storage\\_request\\_t](#).

### Data Fields

```

    • struct {
        uint32_t a
    } val

    • struct {
        void * base
        uint32_t len
    } mem

```

## 6.49.2 Field Documentation

6.49.2.1 `uint32_t te_storage_param_t::a`

6.49.2.2 `struct { ... } te_storage_param_t::val`

6.49.2.3 `void* te_storage_param_t::base`

6.49.2.4 `uint32_t te_storage_param_t::len`



6.49.2.5 struct { ... } te\_storage\_param\_t::mem

The documentation for this union was generated from the following file:

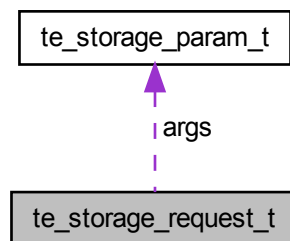
- [ote\\_storage.h](#)

## 6.50 te\_storage\_request\_t Struct Reference

### 6.50.1 Detailed Description

Holds data that the storage service uses in storage requests exchanged with the secure kernel.

Collaboration diagram for te\_storage\_request\_t:



### Data Fields

- `int32_t` [type](#)
- `te_storage_param_t` [args](#) [`TE_MAX_STORAGE_REQUEST_PARAMS`]
- `int32_t` [result](#)

### 6.50.2 Field Documentation

6.50.2.1 `int32_t` `te_storage_request_t::type`

Holds the command.

6.50.2.2 `te_storage_param_t` `te_storage_request_t::args`[`TE_MAX_STORAGE_REQUEST_PARAMS`]

6.50.2.3 `int32_t` `te_storage_request_t::result`

Holds the result.

The documentation for this struct was generated from the following file:

- [ote\\_storage.h](#)

## 6.51 `te_system_info_args_t` Struct Reference

### Data Fields

- `uint32_t si_type`

### 6.51.1 Field Documentation

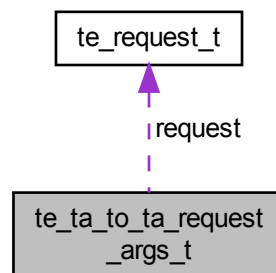
#### 6.51.1.1 `uint32_t te_system_info_args_t::si_type`

The documentation for this struct was generated from the following file:

- [ote\\_task\\_load.h](#)

## 6.52 `te_ta_to_ta_request_args_t` Struct Reference

Collaboration diagram for `te_ta_to_ta_request_args_t`:



### Data Fields

- `te_request_t request`

### 6.52.1 Field Documentation

#### 6.52.1.1 `te_request_t te_ta_to_ta_request_args_t::request`

The documentation for this struct was generated from the following file:

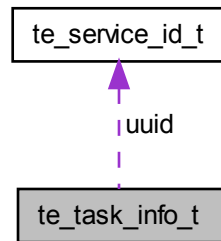
- [ote\\_service.h](#)

## 6.53 `te_task_info_t` Struct Reference

### 6.53.1 Detailed Description

This is compatible with `task_info_t`, field values extracted from manifest record

Collaboration diagram for te\_task\_info\_t:



## Data Fields

- [te\\_service\\_id\\_t](#) `uuid`
- `u_int` `manifest_exists`
- `u_int` `multi_instance`
- `u_int` `min_stack_size`
- `u_int` `min_heap_size`
- `u_int` `map_io_mem_cnt`
- `u_int` `restrict_access`
- `u_int` `authorizations`
- `u_int` `initial_state`
- `char` `task_name` [`OTE_TASK_NAME_MAX_LENGTH`]
- `unsigned char` `task_private_data` [`OTE_TASK_PRIVATE_DATA_LENGTH`]

## 6.53.2 Field Documentation

6.53.2.1 `te_service_id_t te_task_info_t::uuid`

6.53.2.2 `u_int te_task_info_t::manifest_exists`

6.53.2.3 `u_int te_task_info_t::multi_instance`

6.53.2.4 `u_int te_task_info_t::min_stack_size`

6.53.2.5 `u_int te_task_info_t::min_heap_size`

6.53.2.6 `u_int te_task_info_t::map_io_mem_cnt`

6.53.2.7 `u_int te_task_info_t::restrict_access`

6.53.2.8 `u_int te_task_info_t::authorizations`

6.53.2.9 `u_int te_task_info_t::initial_state`

6.53.2.10 `char te_task_info_t::task_name[OTE_TASK_NAME_MAX_LENGTH]`



- 6.54.2.3 `te_app_prepare_args_t` `te_task_request_args_s::ia_prepare`
- 6.54.2.4 `te_get_pending_map_args_t` `te_task_request_args_s::ia_pending_mapping`
- 6.54.2.5 `te_app_start_args_t` `te_task_request_args_s::ia_start`
- 6.54.2.6 `te_app_list_args_t` `te_task_request_args_s::ia_list`
- 6.54.2.7 `te_get_task_info_t` `te_task_request_args_s::ia_get_task_info`
- 6.54.2.8 `te_get_task_mapping_t` `te_task_request_args_s::ia_get_task_mapping`
- 6.54.2.9 `te_app_unload_args_t` `te_task_request_args_s::ia_app_unload`
- 6.54.2.10 `te_app_block_args_t` `te_task_request_args_s::ia_app_block`
- 6.54.2.11 `te_system_info_args_t` `te_task_request_args_s::ia_system_info`
- 6.54.2.12 `union { ... }`

The documentation for this struct was generated from the following file:

- [ote\\_task\\_load.h](#)

## 6.55 `te_task_restrictions_t` Struct Reference

### 6.55.1 Detailed Description

Holds task restrictions requested by the installer (manifest overrides).

#### Note

This does not currently allow to "override" all manifest fields, e.g. the UUID which means that tasks without manifests can not be loaded.

In general: only non-security critical field value overrides can currently be requested (more/all fields can be added if required).

If the task manifest is flagged immutable, TLK will not modify any manifest values by the override mechanism. In this case the installer may choose to reject immutable tasks if overrides are mandatory (any override values will be ignored in this case). For immutable tasks the `APP_RESTRICTIONS` parameter must be set to `NULL`.

This record is compatible with `task_restrictions_t`

### Data Fields

- `uint32_t` [min\\_stack\\_size](#)
- `uint32_t` [min\\_heap\\_size](#)
- `char` [task\\_name](#) [`OTE_TASK_NAME_MAX_LENGTH`]

### 6.55.2 Field Documentation

6.55.2.1 `uint32_t` `te_task_restrictions_t::min_stack_size`

6.55.2.2 `uint32_t` `te_task_restrictions_t::min_heap_size`

### 6.55.2.3 `char te_task_restrictions_t::task_name[OTE_TASK_NAME_MAX_LENGTH]`

The documentation for this struct was generated from the following file:

- [ote\\_task\\_load.h](#)

## 6.56 `te_v_to_p_args_t` Struct Reference

### 6.56.1 Detailed Description

Holds a pointer to the physical address for a specific virtual address.

Used with the [OTE\\_IOCTL\\_V\\_TO\\_P](#) operation.

#### Data Fields

- `void * vaddr`  
*Holds a pointer to the virtual address.*
- `uint64_t paddr`  
*Holds a pointer to the corresponding physical address.*

### 6.56.2 Field Documentation

#### 6.56.2.1 `void* te_v_to_p_args_t::vaddr`

Holds a pointer to the virtual address.

#### 6.56.2.2 `uint64_t te_v_to_p_args_t::paddr`

Holds a pointer to the corresponding physical address.

The documentation for this struct was generated from the following file:

- [ote\\_ext\\_nv.h](#)

## Chapter 7

# File Documentation

### 7.1 nvtlkoss.txt File Reference

### 7.2 ote\_attrs.h File Reference

#### 7.2.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Service Attributes Description:** Declares the service attributes in the TLK.

#### Data Structures

- struct [te\\_attribute\\_t](#)

#### Macros

- #define [OTE\\_ATTR\\_VAL](#) 1 << 29
- #define [OTE\\_ATTR\\_PUB](#) 1 << 28

## Enumerations

- enum `te_attribute_id_t` {
  - `OTE_ATTR_SECRET_VALUE` = 0xC0000000,
  - `OTE_ATTR_RSA_MODULES` = 0xD0000130,
  - `OTE_ATTR_RSA_PUBLIC_EXPONENT` = 0xD0000230,
  - `OTE_ATTR_RSA_PRIVATE_EXPONENT` = 0xC0000330,
  - `OTE_ATTR_RSA_PRIME1` = 0xC0000430,
  - `OTE_ATTR_RSA_PRIME2` = 0xC0000530,
  - `OTE_ATTR_RSA_EXPONENT1` = 0xC0000630,
  - `OTE_ATTR_RSA_EXPONENT2` = 0xC0000730,
  - `OTE_ATTR_RSA_COEFFICIENT` = 0xC0000830,
  - `OTE_ATTR_DSA_PRIME` = 0xD0001031,
  - `OTE_ATTR_DSA_SUBPRIME` = 0xD0001131,
  - `OTE_ATTR_DSA_BASE` = 0xD0001231,
  - `OTE_ATTR_DSA_PUBLIC_VALUE` = 0xD0000131,
  - `OTE_ATTR_DSA_PRIVATE_VALUE` = 0xD0000231,
  - `OTE_ATTR_DH_PRIME` = 0xD0001032,
  - `OTE_ATTR_DH_SUBPRIME` = 0xD0001132,
  - `OTE_ATTR_DH_BASE` = 0xD0001232,
  - `OTE_ATTR_DH_X_BITS` = 0xF0001332,
  - `OTE_ATTR_DH_PUBLIC_VALUE` = 0xD0000132,
  - `OTE_ATTR_DH_PRIVATE_VALUE` = 0xC0000232,
  - `OTE_ATTR_RSA_OAEP_LABEL` = 0xD0000930,
  - `OTE_ATTR_RSA_PSS_SALT_LENGTH` = 0xF0000A30 }

## Functions

- `te_error_t te_set_mem_attribute` (`te_attribute_t` \*attr, `te_attribute_id_t` id, void \*buffer, uint32\_t size)
- `te_error_t te_get_mem_attribute_buffer` (`te_attribute_t` \*attr, void \*\*ret)
- `te_error_t te_get_mem_attribute_size` (`te_attribute_t` \*attr, size\_t \*ret)
- void `te_copy_mem_attribute` (void \*buffer, `te_attribute_t` \*key)
- `te_error_t te_set_int_attribute` (`te_attribute_t` \*attr, `te_attribute_id_t` id, uint32\_t a, uint32\_t b)
- void `te_free_internal_attribute` (`te_attribute_t` \*attr)
- `te_error_t te_copy_attribute` (`te_attribute_t` \*dst, `te_attribute_t` \*src)

## 7.3 ote\_client.h File Reference

### 7.3.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Client Communications Description:** Declares the TLK client interface.

## Data Structures

- struct `te_answer`
- struct `ote_opensession`
  - Opens an open trusted environment (OTE) session.*
- struct `ote_closesession`
  - Closes an OTE session.*
- struct `ote_launchop`
  - Launches an operation request.*
- union `te_cmd`
- struct `ote_file_create_params_t`



- struct [ote\\_file\\_delete\\_params\\_t](#)
- struct [ote\\_file\\_open\\_params\\_t](#)
- struct [ote\\_file\\_close\\_params\\_t](#)
- struct [ote\\_file\\_write\\_params\\_t](#)
- struct [ote\\_file\\_read\\_params\\_t](#)
- struct [ote\\_file\\_seek\\_params\\_t](#)
- struct [ote\\_file\\_trunc\\_params\\_t](#)
- struct [ote\\_file\\_get\\_size\\_params\\_t](#)
- struct [ote\\_file\\_rpmb\\_write\\_params\\_t](#)
- struct [ote\\_file\\_rpmb\\_read\\_params\\_t](#)
- union [ote\\_file\\_req\\_params\\_t](#)
- struct [ote\\_file\\_req\\_t](#)
- struct [ote\\_ss\\_op\\_t](#)

## Macros

- #define [TLK\\_DEVICE\\_BASE\\_NAME](#) "tlk\_device"
- #define [TE\\_IOCTL\\_MAGIC\\_NUMBER](#) ('t')
- #define [TE\\_IOCTL\\_OPEN\\_CLIENT\\_SESSION](#) \_IOWR([TE\\_IOCTL\\_MAGIC\\_NUMBER](#), 0x10, union [te\\_cmd](#))
- #define [TE\\_IOCTL\\_CLOSE\\_CLIENT\\_SESSION](#) \_IOWR([TE\\_IOCTL\\_MAGIC\\_NUMBER](#), 0x11, union [te\\_cmd](#))
- #define [TE\\_IOCTL\\_LAUNCH\\_OP](#) \_IOWR([TE\\_IOCTL\\_MAGIC\\_NUMBER](#), 0x14, union [te\\_cmd](#))
- #define [TE\\_IOCTL\\_SS\\_NEW\\_REQ](#) \_IOR([TE\\_IOCTL\\_MAGIC\\_NUMBER](#), 0x20, [ote\\_ss\\_op\\_t](#))
- #define [TE\\_IOCTL\\_SS\\_REQ\\_COMPLETE](#) \_IOWR([TE\\_IOCTL\\_MAGIC\\_NUMBER](#), 0x21, [ote\\_ss\\_op\\_t](#))
- #define [OTE\\_MAX\\_DIR\\_NAME\\_LEN](#) (64)
- #define [OTE\\_MAX\\_FILE\\_NAME\\_LEN](#) (128)
- #define [OTE\\_MAX\\_DATA\\_SIZE](#) (2048)
- #define [OTE\\_RPMB\\_FRAME\\_SIZE](#) 512
- #define [SS\\_OP\\_MAX\\_DATA\\_SIZE](#) 0x1000

## Enumerations

- enum {  
[TLK\\_SMC\\_REQUEST](#) = 0xFFFF1000,  
[TLK\\_SMC\\_GET\\_MORE](#) = 0xFFFF1001,  
[TLK\\_SMC\\_ANSWER](#) = 0xFFFF1002,  
[TLK\\_SMC\\_NO\\_ANSWER](#) = 0xFFFF1003,  
[TLK\\_SMC\\_OPEN\\_SESSION](#) = 0xFFFF1004,  
[TLK\\_SMC\\_CLOSE\\_SESSION](#) = 0xFFFF1005 }

*Defines secure monitor calls (SMC) that clients use to communicate with trusted applications (TAs) in the secure world.*

- enum {  
[OTE\\_FILE\\_REQ\\_TYPE\\_CREATE](#) = 0x1,  
[OTE\\_FILE\\_REQ\\_TYPE\\_DELETE](#) = 0x2,  
[OTE\\_FILE\\_REQ\\_TYPE\\_OPEN](#) = 0x3,  
[OTE\\_FILE\\_REQ\\_TYPE\\_CLOSE](#) = 0x4,  
[OTE\\_FILE\\_REQ\\_TYPE\\_READ](#) = 0x5,  
[OTE\\_FILE\\_REQ\\_TYPE\\_WRITE](#) = 0x6,  
[OTE\\_FILE\\_REQ\\_TYPE\\_GET\\_SIZE](#) = 0x7,  
[OTE\\_FILE\\_REQ\\_TYPE\\_SEEK](#) = 0x8,  
[OTE\\_FILE\\_REQ\\_TYPE\\_TRUNC](#) = 0x9,  
[OTE\\_FILE\\_REQ\\_TYPE\\_RPMB\\_WRITE](#) = 0x1001,  
[OTE\\_FILE\\_REQ\\_TYPE\\_RPMB\\_READ](#) = 0x1002 }

- enum {  
[OTE\\_FILE\\_REQ\\_FLAGS\\_ACCESS\\_RO](#) = 1,  
[OTE\\_FILE\\_REQ\\_FLAGS\\_ACCESS\\_WO](#) = 2,  
[OTE\\_FILE\\_REQ\\_FLAGS\\_ACCESS\\_RW](#) = 3 }
- enum {  
[OTE\\_SEEK\\_WHENCE\\_SET](#) = 1,  
[OTE\\_SEEK\\_WHENCE\\_CUR](#) = 2,  
[OTE\\_SEEK\\_WHENCE\\_END](#) = 3 }

## 7.4 ote\_command.h File Reference

### 7.4.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Common Commands Description:** Declares the common commands in the TLK interface.

#### Functions

- [te\\_error\\_t te\\_open\\_session](#) ([te\\_session\\_t](#) \*session, [te\\_service\\_id\\_t](#) \*service, [te\\_operation\\_t](#) \*operation)
- void [te\\_close\\_session](#) ([te\\_session\\_t](#) \*session)
- [te\\_operation\\_t](#) \* [te\\_create\\_operation](#) (void)
- void [te\\_init\\_operation](#) ([te\\_operation\\_t](#) \*te\_op)
- void [te\\_deinit\\_operation](#) ([te\\_operation\\_t](#) \*teOp)
- [te\\_error\\_t](#) [te\\_launch\\_operation](#) ([te\\_session\\_t](#) \*session, [te\\_operation\\_t](#) \*te\_op)
- void [te\\_oper\\_set\\_command](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t command)
- void [te\\_oper\\_set\\_param\\_int\\_ro](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, uint32\_t lnt)
- void [te\\_oper\\_set\\_param\\_int\\_rw](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, uint32\_t lnt)
- void [te\\_oper\\_set\\_param\\_mem\\_ro](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, void \*base, uint32\_t len)
- void [te\\_oper\\_set\\_param\\_mem\\_rw](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, void \*base, uint32\_t len)
- uint32\_t [te\\_oper\\_get\\_command](#) ([te\\_operation\\_t](#) \*te\_op)
- uint32\_t [te\\_oper\\_get\\_num\\_params](#) ([te\\_operation\\_t](#) \*te\_op)
- [te\\_error\\_t](#) [te\\_oper\\_get\\_param\\_type](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, [te\\_oper\\_param\\_type\\_t](#) \*type)
- [te\\_error\\_t](#) [te\\_oper\\_get\\_param\\_int](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, uint32\_t \*lnt)
- [te\\_error\\_t](#) [te\\_oper\\_get\\_param\\_mem](#) ([te\\_operation\\_t](#) \*te\_op, uint32\_t index, void \*\*base, uint32\_t \*len)
- void [te\\_operation\\_reset](#) ([te\\_operation\\_t](#) \*te\_op)

## 7.5 ote\_common.h File Reference

### 7.5.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Common Declarations Description:** Declares the common declarations in the TLK interface.

#### Data Structures

- struct [te\\_service\\_id\\_t](#)
- union [te\\_session\\_t](#)
- struct [te\\_oper\\_param\\_t](#)
- struct [te\\_operation\\_t](#)

## Macros

- `#define OTE_TASK_NAME_MAX_LENGTH 24`
- `#define OTE_TASK_PRIVATE_DATA_LENGTH 20`

## Typedefs

- `typedef uint64_t cmnptr_t`

## Enumerations

- `enum te_oper_param_type_t {  
 TE_PARAM_TYPE_NONE = 0,  
 TE_PARAM_TYPE_INT_RO = 1,  
 TE_PARAM_TYPE_INT_RW = 2,  
 TE_PARAM_TYPE_MEM_RO = 3,  
 TE_PARAM_TYPE_MEM_RW = 4 }`

## Functions

- `te_result_origin_t te_get_result_origin (te_session_t *session)`

## 7.6 ote\_crypto.h File Reference

### 7.6.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Cryptography Description:** Declares the cryptography APIs in the TLK.

## Data Structures

- `struct te_crypto_operation_info_t`
- `struct __te_crypto_operation_t`
- `struct te_crypto_rsa_key_t`

## Typedefs

- `typedef struct __te_crypto_object * te_crypto_object_t`
- `typedef struct  
 __te_crypto_operation_t * te_crypto_operation_t`

## Enumerations

- enum `te_oper_crypto_algo_t` {  
`OTE_ALG_AES_ECB_NOPAD` = 0x10000010,  
`OTE_ALG_AES_CBC_NOPAD` = 0x10000110,  
`OTE_ALG_AES_CTR` = 0x10000210,  
`OTE_ALG_AES_CTS` = 0x10000310,  
`OTE_ALG_AES_ECB` = 0x10000510,  
`OTE_ALG_AES_CBC` = 0x10000610,  
`OTE_ALG_AES_CMAC_128` = 0x20000110,  
`OTE_ALG_AES_CMAC_192` = 0x20000120,  
`OTE_ALG_AES_CMAC_256` = 0x20000130,  
`OTE_ALG_SHA_HMAC_224` = 0x20000210,  
`OTE_ALG_SHA_HMAC_256` = 0x20000220,  
`OTE_ALG_SHA_HMAC_384` = 0x20000230,  
`OTE_ALG_SHA_HMAC_512` = 0x20000240,  
`OTE_ALG_RSA_PKCS_OAEP` = 0x30000100,  
`OTE_ALG_RSA_PSS` = 0x30000200 }
- enum `te_oper_crypto_algo_mode_t` {  
`OTE_ALG_MODE_ENCRYPT`,  
`OTE_ALG_MODE_DECRYPT`,  
`OTE_ALG_MODE_SIGN`,  
`OTE_ALG_MODE_VERIFY`,  
`OTE_ALG_MODE_DIGEST`,  
`OTE_ALG_MODE_DERIVE` }

## Functions

- `te_error_t te_allocate_object (te_crypto_object_t *obj)`
- `te_error_t te_populate_object (te_crypto_object_t obj, te_attribute_t *attrs, uint32_t attr_count)`
- `void te_free_object (te_crypto_object_t obj)`
- `te_error_t te_allocate_operation (te_crypto_operation_t *oper, te_oper_crypto_algo_t algorithm, te_oper_crypto_algo_mode_t mode)`
- `te_error_t te_set_operation_key (te_crypto_operation_t oper, te_crypto_object_t obj)`
- `te_error_t te_cipher_init (te_crypto_operation_t oper, void *iv, uint32_t iv_size)`
- `te_error_t te_cipher_update (te_crypto_operation_t oper, void *src_data, uint32_t src_size, void *dst_data, uint32_t *dst_size)`
- `te_error_t te_cipher_do_final (te_crypto_operation_t oper, void *src_data, uint32_t src_len, void *dst_data, uint32_t *dst_len)`
- `te_error_t te_rsa_init (te_crypto_operation_t oper)`
- `te_error_t te_rsa_handle_request (te_crypto_operation_t oper, void *src_data, uint32_t src_size, void *dst_data, uint32_t *dst_size)`
- `void te_free_operation (te_crypto_operation_t oper)`
- `void te_generate_random (void *buffer, size_t size)`
- `te_error_t te_get_attribute_by_id (te_crypto_object_t object, te_attribute_id_t id, te_attribute_t **ret)`

## 7.7 ote\_error.h File Reference

### 7.7.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Error Handling Description:** Declares the error handling codes for the TLK interface.

## Enumerations

- enum [te\\_error\\_t](#) {  
[OTE\\_SUCCESS](#) = 0x00000000,  
[OTE\\_ERROR\\_NO\\_ERROR](#) = 0x00000000,  
[OTE\\_ERROR\\_GENERIC](#) = 0xFFFF0000,  
[OTE\\_ERROR\\_ACCESS\\_DENIED](#) = 0xFFFF0001,  
[OTE\\_ERROR\\_CANCEL](#) = 0xFFFF0002,  
[OTE\\_ERROR\\_ACCESS\\_CONFLICT](#) = 0xFFFF0003,  
[OTE\\_ERROR\\_EXCESS\\_DATA](#) = 0xFFFF0004,  
[OTE\\_ERROR\\_BAD\\_FORMAT](#) = 0xFFFF0005,  
[OTE\\_ERROR\\_BAD\\_PARAMETERS](#) = 0xFFFF0006,  
[OTE\\_ERROR\\_BAD\\_STATE](#) = 0xFFFF0007,  
[OTE\\_ERROR\\_ITEM\\_NOT\\_FOUND](#) = 0xFFFF0008,  
[OTE\\_ERROR\\_NOT\\_IMPLEMENTED](#) = 0xFFFF0009,  
[OTE\\_ERROR\\_NOT\\_SUPPORTED](#) = 0xFFFF000A,  
[OTE\\_ERROR\\_NO\\_DATA](#) = 0xFFFF000B,  
[OTE\\_ERROR\\_OUT\\_OF\\_MEMORY](#) = 0xFFFF000C,  
[OTE\\_ERROR\\_BUSY](#) = 0xFFFF000D,  
[OTE\\_ERROR\\_COMMUNICATION](#) = 0xFFFF000E,  
[OTE\\_ERROR\\_SECURITY](#) = 0xFFFF000F,  
[OTE\\_ERROR\\_SHORT\\_BUFFER](#) = 0xFFFF0010,  
[OTE\\_ERROR\\_BLOCKED](#) = 0xFFFF0011,  
[OTE\\_ERROR\\_NO\\_ANSWER](#) = 0xFFFF1003 }  
*Defines Open Trusted Environment (OTE) error codes.*

- enum [te\\_result\\_origin\\_t](#) {  
[OTE\\_RESULT\\_ORIGIN\\_API](#) = 1,  
[OTE\\_RESULT\\_ORIGIN\\_COMMS](#) = 2,  
[OTE\\_RESULT\\_ORIGIN\\_KERNEL](#) = 3,  
[OTE\\_RESULT\\_ORIGIN\\_TRUSTED\\_APP](#) = 4 }  
*Defines the origin of an error.*

## 7.8 ote\_ext\_nv.h File Reference

### 7.8.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Memory/Cache Management Description:** Declares memory/cache management in the TLK.

## Data Structures

- struct [te\\_map\\_mem\\_addr\\_args\\_t](#)  
*Holds a pointer to the map memory for a specific [OTE\\_CONFIG\\_MAP\\_MEM](#) ID value.*
- struct [te\\_v\\_to\\_p\\_args\\_t](#)  
*Holds a pointer to the physical address for a specific virtual address.*
- struct [te\\_cache\\_maint\\_args\\_t](#)  
*Holds an op code and data used to for cache maintenance.*

## Enumerations

- enum {  
[OTE\\_IOCTL\\_GET\\_MAP\\_MEM\\_ADDR](#) = 1,  
[OTE\\_IOCTL\\_V\\_TO\\_P](#) = 2,  
[OTE\\_IOCTL\\_CACHE\\_MAINT](#) = 3 }

- enum `te_ext_nv_cache_maint_op_t` {  
`OTE_EXT_NV_CM_OP_CLEAN` = 1,  
`OTE_EXT_NV_CM_OP_INVALIDATE` = 2,  
`OTE_EXT_NV_CM_OP_FLUSH` = 3 }

## Functions

- `te_error_t te_ext_nv_cache_maint` (`te_ext_nv_cache_maint_op_t` op, void \*addr, uint32\_t length)
- `te_error_t te_ext_nv_virt_to_phys` (void \*addr, uint64\_t \*paddr)
- `te_error_t te_ext_nv_get_map_addr` (uint32\_t id, void \*\*addr)

## 7.9 ote\_manifest.h File Reference

### 7.9.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Manifest Layout Description:** Declares the layout of the manifest in the TLK.

## Data Structures

- struct `OTE_MANIFEST`

## Macros

- `#define OTE_CONFIG_MIN_STACK_SIZE`(sz) `OTE_CONFIG_KEY_MIN_STACK_SIZE`, sz
- `#define OTE_CONFIG_MIN_HEAP_SIZE`(sz) `OTE_CONFIG_KEY_MIN_HEAP_SIZE`, sz
- `#define OTE_CONFIG_MAP_MEM`(id, off, sz) `OTE_CONFIG_KEY_MAP_MEM`, id, off, sz
- `#define OTE_CONFIG_RESTRICT_ACCESS`(clients) `OTE_CONFIG_KEY_RESTRICT_ACCESS`, clients
- `#define OTE_CONFIG_AUTHORIZE`(perm) `OTE_CONFIG_KEY_AUTHORIZE`, perm
- `#define OTE_CONFIG_TASK_INITIAL_STATE`(state) `OTE_CONFIG_KEY_TASK_ISTATE`, state
- `#define OTE_MANIFEST_ATTRS` `__attribute__((aligned(4))) __attribute__((section(".ote.manifest")))`

## Enumerations

- enum `ote_config_key_t` {  
`OTE_CONFIG_KEY_MIN_STACK_SIZE` = 1,  
`OTE_CONFIG_KEY_MIN_HEAP_SIZE` = 2,  
`OTE_CONFIG_KEY_MAP_MEM` = 3,  
`OTE_CONFIG_KEY_RESTRICT_ACCESS` = 4,  
`OTE_CONFIG_KEY_AUTHORIZE` = 5,  
`OTE_CONFIG_KEY_TASK_ISTATE` = 6 }
- enum {  
`OTE_RESTRICT_SECURE_TASKS` = 1 << 0,  
`OTE_RESTRICT_NON_SECURE_APPS` = 1 << 1 }
- enum { `OTE_AUTHORIZE_INSTALL` = 1 << 10 }
- enum {  
`OTE_MANIFEST_TASK_ISTATE_IMMUTABLE` = 1 << 0,  
`OTE_MANIFEST_TASK_ISTATE_STICKY` = 1 << 1,  
`OTE_MANIFEST_TASK_ISTATE_BLOCKED` = 1 << 2 }

## 7.10 ote\_memory.h File Reference

### 7.10.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Memory Functions Description:** Declares the memory functions in the TLK.

#### Functions

- void \* [te\\_mem\\_alloc](#) (uint32\_t size)
- void \* [te\\_mem\\_calloc](#) (uint32\_t size)
- void [te\\_mem\\_free](#) (void \*buffer)
- void [te\\_mem\\_fill](#) (void \*buffer, uint32\_t value, uint32\_t size)
- void [te\\_mem\\_move](#) (void \*dest, void \*src, uint32\_t size)
- int [te\\_mem\\_compare](#) (void \*buffer1, void \*buffer2, uint32\_t size)

## 7.11 ote\_nvcrypto.h File Reference

### 7.11.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: NVIDIA Cryptography Description:** Declares the cryptography APIs in the TLK.

#### Functions

- [te\\_error\\_t ote\\_nvcrypto\\_init](#) (void)
- [te\\_error\\_t ote\\_nvcrypto\\_deinit](#) (void)
- [te\\_error\\_t ote\\_nvcrypto\\_get\\_keybox](#) (uint32\_t keybox\_index, void \*buf, uint32\_t \*len)
- [te\\_error\\_t ote\\_nvcrypto\\_get\\_storage\\_key](#) (uint8\_t \*key, uint32\_t key\_size)

*Gets the storage key.*

## 7.12 ote\_otf.h File Reference

### 7.12.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: On-the-Fly (OTF) Decoder Service Description:** Declares functions for the TLK OTF decoder service.

#### Functions

- [te\\_error\\_t ote\\_otf\\_init](#) ([te\\_session\\_t](#) \*\*otfSession)  
*Initializes the on-the-fly (OTF) hardware.*
- [te\\_error\\_t ote\\_otf\\_deinit](#) ([te\\_session\\_t](#) \*\*otfSession)  
*Resets the OTF hardware and erases any previous keys.*
- [te\\_error\\_t ote\\_otf\\_setkey](#) (void \*buffer, uint32\_t len, uint32\_t \*keySlot, [te\\_session\\_t](#) \*otfSession)  
*Sets the key to be used by the OTF hardware.*
- [te\\_error\\_t ote\\_otf\\_erasekey](#) ([te\\_session\\_t](#) \*otfSession)  
*Erases keys from the OTF hardware.*

## 7.13 ote\_rtc.h File Reference

### 7.13.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Real-Time Clock (RTC) Services Description:** Declares common declarations and functions for the TLK RTC service.

#### Functions

- [te\\_error\\_t ote\\_rtc\\_init](#) (void)  
*Initializes the RTC hardware.*
- [te\\_error\\_t ote\\_rtc\\_deinit](#) (void)  
*Resets the RTC hardware and erases any previous keys.*
- [te\\_error\\_t ote\\_rtc\\_get\\_time](#) (uint32\_t \*rtc)  
*Gets the RTC from the RTC hardware.*

## 7.14 ote\_service.h File Reference

### 7.14.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Service Interface Description:** Declares data types and functions for the TLK services.

#### Data Structures

- struct [te\\_request\\_t](#)  
*Holds the layout of the [te\\_oper\\_param\\_t](#) structures which must match the layout sent in by the non-secure (NS) world via the TrustZone Secure Monitor Call (TZ SMC) path.*
- struct [te\\_ta\\_to\\_ta\\_request\\_args\\_t](#)
- struct [te\\_entry\\_point\\_message\\_t](#)
- struct [te\\_identity\\_t](#)
- struct [te\\_get\\_property\\_args\\_t](#)
- struct [te\\_device\\_unique\\_id](#)

#### Macros

- [#define DEVICE\\_UID\\_SIZE\\_BYTES](#) 16
- [#define OTE\\_TE\\_FPRINTF\\_PREFIX\\_MAX\\_LENGTH](#) (OTE\_TASK\_NAME\_MAX\_LENGTH + 4)

#### Enumerations

- enum {  
    [TE\\_CRITICAL](#) = 0,  
    [TE\\_ERR](#) = 1,  
    [TE\\_INFO](#) = 2,  
    [TE\\_SECURE](#) = 3,  
    [TE\\_INTERFACE](#) = 4,  
    [TE\\_RESULT](#) = 5 }



- enum {  
[CREATE\\_INSTANCE](#) = 1UL,  
[DESTROY\\_INSTANCE](#) = 2UL,  
[OPEN\\_SESSION](#) = 3UL,  
[CLOSE\\_SESSION](#) = 4UL,  
[LAUNCH\\_OPERATION](#) = 5UL }
- enum {  
[OTE\\_IOCTL\\_TA\\_TO\\_TA\\_REQUEST](#) = 4UL,  
[OTE\\_IOCTL\\_GET\\_PROPERTY](#) = 5UL,  
[OTE\\_IOCTL\\_GET\\_DEVICE\\_ID](#) = 6UL,  
[OTE\\_IOCTL\\_GET\\_TIME\\_US](#) = 7UL,  
[OTE\\_IOCTL\\_GET\\_RAND32](#) = 8UL,  
[OTE\\_IOCTL\\_SS\\_REQUEST](#) = 9UL,  
[OTE\\_IOCTL\\_TASK\\_REQUEST](#) = 10UL }  
*Defines IOCTL syscall interface parameters.*
- enum {  
[TE\\_PROP\\_DATA\\_TYPE\\_UUID](#) = 1,  
[TE\\_PROP\\_DATA\\_TYPE\\_IDENTITY](#) = 2 }
- enum [te\\_property\\_type\\_t](#) {  
[TE\\_PROPERTY\\_CURRENT\\_TA](#) = 0xFFFFFFFF,  
[TE\\_PROPERTY\\_CURRENT\\_CLIENT](#) = 0xFFFFFFFFE,  
[TE\\_PROPERTY\\_TE\\_IMPLEMENTATION](#) = 0xFFFFFFFFD }

## Functions

- void [te\\_exit\\_service](#) (void)
- [te\\_error\\_t](#) [te\\_init](#) (int argc, char \*\*argv)
- void [te\\_destroy](#) (void)
- [te\\_error\\_t](#) [te\\_create\\_instance\\_iface](#) (void)
- void [te\\_destroy\\_instance\\_iface](#) (void)
- [te\\_error\\_t](#) [te\\_open\\_session\\_iface](#) (void \*\*sctx, [te\\_operation\\_t](#) \*oper)
- void [te\\_close\\_session\\_iface](#) (void \*sctx)
- [te\\_error\\_t](#) [te\\_receive\\_operation\\_iface](#) (void \*sctx, [te\\_operation\\_t](#) \*oper)
- void \* [ote\\_get\\_instance\\_data](#) (void)
- void [ote\\_set\\_instance\\_data](#) (void \*sessionContext)
- [te\\_error\\_t](#) [te\\_get\\_current\\_ta\\_uuid](#) ([te\\_service\\_id\\_t](#) \*value)
- [te\\_error\\_t](#) [te\\_get\\_client\\_ta\\_identity](#) ([te\\_identity\\_t](#) \*value)
- [te\\_error\\_t](#) [te\\_get\\_device\\_unique\\_id](#) ([te\\_device\\_unique\\_id](#) \*uid)
- void [te\\_fprintf\\_set\\_prefix](#) (const char \*prefix)
- int [te\\_fprintf](#) (int fd, char \*fmt,...) \_\_PRINTFLIKE(2)
- int void [te\\_oper\\_dump\\_param](#) ([te\\_oper\\_param\\_t](#) \*param)
- void [te\\_oper\\_dump\\_param\\_list](#) ([te\\_operation\\_t](#) \*te\_op)

## 7.15 ote\_storage.h File Reference

### 7.15.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Storage Services Description:** Declares data types and functions for TLK storage services.

### Data Structures

- union [te\\_storage\\_param\\_t](#)
- struct [te\\_storage\\_request\\_t](#)

## Macros

- `#define TE_STORAGE_OBJID_MAX_LEN 64`
- `#define TE_MAX_STORAGE_REQUEST_PARAMS 4`

## Typedefs

- `typedef struct`  
`__te_storage_object * te\_storage\_object\_t`

## Enumerations

- `enum {`  
`OTE_FILE_REQ_TYPE_CREATE = 0x1,`  
`OTE_FILE_REQ_TYPE_DELETE = 0x2,`  
`OTE_FILE_REQ_TYPE_OPEN = 0x3,`  
`OTE_FILE_REQ_TYPE_CLOSE = 0x4,`  
`OTE_FILE_REQ_TYPE_READ = 0x5,`  
`OTE_FILE_REQ_TYPE_WRITE = 0x6,`  
`OTE_FILE_REQ_TYPE_GET_SIZE = 0x7,`  
`OTE_FILE_REQ_TYPE_SEEK = 0x8,`  
`OTE_FILE_REQ_TYPE_TRUNC = 0x9,`  
`OTE_FILE_REQ_TYPE_RPMB_WRITE = 0x1001,`  
`OTE_FILE_REQ_TYPE_RPMB_READ = 0x1002 }`
- `enum te\_storage\_flags\_t {`  
`OTE_STORAGE_FLAG_ACCESS_READ = 0x1,`  
`OTE_STORAGE_FLAG_ACCESS_WRITE = 0x2,`  
`OTE_STORAGE_FLAG_ACCESS_WRITE_META = 0x4 }`
- `enum te\_storage\_whence\_t {`  
`OTE_STORAGE_SEEK_WHENCE_SET = 0x1,`  
`OTE_STORAGE_SEEK_WHENCE_CUR = 0x2,`  
`OTE_STORAGE_SEEK_WHENCE_END = 0x3 }`

## Functions

- `te\_error\_t te\_create\_storage\_object (char *name, te\_storage\_flags\_t flags, te\_storage\_object\_t *obj)`
- `te\_error\_t te\_open\_storage\_object (char *name, te\_storage\_flags\_t flags, te\_storage\_object\_t *obj)`
- `te\_error\_t te\_read\_storage\_object (te\_storage\_object\_t obj, void *buffer, uint32\_t size, uint32\_t *count)`
- `te\_error\_t te\_write\_storage\_object (te\_storage\_object\_t obj, void *buffer, uint32\_t size)`
- `te\_error\_t te\_get\_storage\_object\_size (te\_storage\_object\_t obj, uint32\_t *size)`
- `te\_error\_t te\_seek\_storage\_object (te\_storage\_object\_t obj, int32\_t offset, te\_storage\_whence\_t whence)`
- `te\_error\_t te\_trunc\_storage\_object (te\_storage\_object\_t obj, uint32\_t size)`
- `te\_error\_t te\_delete\_storage\_object (te\_storage\_object\_t obj)`
- `te\_error\_t te\_close\_storage\_object (te\_storage\_object\_t obj)`

## 7.16 ote\_task\_load.h File Reference

### 7.16.1 Detailed Description

**NVIDIA Trusted Little Kernel Interface: Task-Loading Interface Description:** Declares data types and functions for the TLK task-loading interface.

## Data Structures

- struct [te\\_app\\_load\\_memory\\_request\\_args\\_t](#)
- struct [te\\_task\\_info\\_t](#)
- struct [te\\_app\\_prepare\\_args\\_t](#)
- struct [te\\_task\\_restrictions\\_t](#)
- struct [te\\_app\\_start\\_args\\_t](#)
- struct [te\\_app\\_list\\_args\\_t](#)

*Holds arguments for the ioctl handler.*

- struct [te\\_list\\_apps](#)
- struct [te\\_get\\_task\\_info\\_t](#)
- struct [te\\_memory\\_mapping\\_t](#)
- struct [te\\_get\\_task\\_mapping\\_t](#)
- struct [te\\_get\\_pending\\_map\\_args\\_t](#)
- struct [te\\_system\\_info\\_args\\_t](#)
- struct [te\\_app\\_unload\\_args\\_t](#)
- struct [te\\_app\\_unload\\_t](#)
- struct [te\\_app\\_block\\_args\\_t](#)
- struct [te\\_app\\_block\\_t](#)
- struct [te\\_task\\_request\\_args\\_s](#)

## Typedefs

- typedef struct [te\\_list\\_apps](#) [te\\_list\\_apps\\_t](#)
- typedef struct [te\\_task\\_request\\_args\\_s](#) [te\\_task\\_request\\_args\\_t](#)

## Enumerations

- enum [te\\_get\\_info\\_type\\_t](#) {  
[OTE\\_GET\\_TASK\\_INFO\\_REQUEST\\_INDEX](#),  
[OTE\\_GET\\_TASK\\_INFO\\_REQUEST\\_UUID](#),  
[OTE\\_GET\\_TASK\\_INFO\\_REQUEST\\_SELF](#) }
- enum [te\\_app\\_id\\_t](#) {  
[OTE\\_APP\\_ID\\_INDEX](#),  
[OTE\\_APP\\_ID\\_UUID](#) }
- enum [te\\_task\\_opcode\\_t](#) {  
[OTE\\_TASK\\_OP\\_UNKNOWN](#),  
[OTE\\_TASK\\_OP\\_MEMORY\\_REQUEST](#),  
[OTE\\_TASK\\_OP\\_PREPARE](#),  
[OTE\\_TASK\\_OP\\_START](#),  
[OTE\\_TASK\\_OP\\_LIST](#),  
[OTE\\_TASK\\_OP\\_GET\\_TASK\\_INFO](#),  
[OTE\\_TASK\\_OP\\_SYSTEM\\_INFO](#),  
[OTE\\_TASK\\_OP\\_PENDING\\_MAPPING](#),  
[OTE\\_TASK\\_OP\\_UNLOAD](#),  
[OTE\\_TASK\\_OP\\_BLOCK](#),  
[OTE\\_TASK\\_OP\\_UNBLOCK](#),  
[OTE\\_TASK\\_OP\\_GET\\_MAPPING](#) }

## Functions

- [te\\_error\\_t te\\_app\\_request\\_memory](#) (u\_int app\_size, uintptr\_t \*app\_addr, uint32\_t \*app\_handle)
- [te\\_error\\_t te\\_app\\_prepare](#) (uint32\_t app\_handle, [te\\_task\\_info\\_t](#) \*app\_task\_info)
- [te\\_error\\_t te\\_app\\_start](#) (uint32\_t app\_handle, uint32\_t app\_reject, [te\\_task\\_restrictions\\_t](#) \*app\_restrictions)
- [te\\_error\\_t te\\_list\\_apps](#) ([te\\_list\\_apps\\_t](#) \*app\_list)
- [te\\_error\\_t te\\_task\\_get\\_info](#) ([te\\_get\\_task\\_info\\_t](#) \*task\_info)
- [te\\_error\\_t te\\_task\\_get\\_mapping](#) ([te\\_get\\_task\\_mapping\\_t](#) \*task\_mapping)
- [te\\_error\\_t te\\_get\\_pending\\_task\\_mapping](#) (uint32\_t app\_handle, [te\\_memory\\_mapping\\_t](#) \*app\_map)
- [te\\_error\\_t te\\_task\\_get\\_name\\_self](#) (char \*name, uint32\_t \*len\_p)
- [te\\_error\\_t te\\_task\\_system\\_info](#) (uint32\_t type)
- [te\\_error\\_t te\\_app\\_unload](#) ([te\\_app\\_unload\\_t](#) \*arg\_unload)
- [te\\_error\\_t te\\_app\\_block](#) ([te\\_app\\_block\\_t](#) \*app)
- [te\\_error\\_t te\\_app\\_unblock](#) ([te\\_app\\_block\\_t](#) \*app)