TSI-X

Reference Manual for Extended TSI

UNIGRAF

Copyright

Notice

The information in this manual has been verified on the date of issue. The authors reserve rights to make any changes to this product and revise the information without obligation to notify any person about such revisions or changes.

Edition

Title	TSI-X Reference Manual
Issue date	29. November. 2019

Company information

Unigraf Oy

Piispantilankuja 4
FI-02240 ESPOO
Finland

Phone. +358 9 589 550

e-mail: info@unigraf.fi
web: http://www.unigraf.fi

Trademarks

Table of Contents

1 GENERAL

1.1 About this document

This document applies to TSI software release 1.10 [R25] (TSI.DLL)

1.1.1 History

- 8.7.2014
  Initial version for evaluation.
- 4.8.2014
  Revised for first release. Added error codes. Added history entry for first release.
- 2.9.2014
  Revised for second release. Added error codes, Added status log function descriptions, Added reference frame capture function description, Added description of test 2, Added descriptions of new configuration items.
- 27.11.2014
  Added Input parameter related functions, Added device parameter related functions, Added load/save reference functions, Added audio graphical preview function.
- 15.1.2015
  Finalized and revised for release 1.2
- 22.1.2015
  Added message log descriptions to tests
- 13.2.2015
  Added V-by-One related configuration item descriptions.
- 30.4.2015
  Added missing configuration item definitions, Revised 3.4.6 TSI_VIN_Enable, 3.4.5 TSI_VIN_Select, Test run example logs updated. Revised and finalized for release 1.2 [R2], Replaced Vx1 short with “V-by-One”, except for defines and references to defines.
- 26.6.2015
  Revised the history section. Updated TSI_TS_RunTest description. Updated Using the TSI API with more detailed information.
- 18.8.2015
  Added descriptions for functions TSI_REP_BeginLogRecord, TSI_REP_EndLogRecord and TSI_STLOG_WaitMessage
- 14.10.2015
  Revised error descriptions; Added TSI_R_INPUT_INTERLACE configuration item; Added TSI_TS_WaitInputSignal, Revised for 1.3 [R6] release.
- 29.6.2016
  Revised error description, Added configuration items TSI_R_INFOFRAME_* and TSI_HDMI_RX_*, Added ARC configuration item.

(Continued...)

(...Continued)

- 24.8.2016
  Revised for TSI 1.6 [R1]. Added PPM file type ID to save reference function, Added DP RX electrical test parameter definitions table, Added description of HDMI and DP electrical tests, Added CEC functionality test, Changed the info frame access interface to be more extensible.

- 24.11.2016
  Revised for TSI 1.6 [R3]. Added CRC Test definitions, Added CRC configuration item definitions.

- 9.12.2016
  Revised for TSI 1.6 [R4]. Removed un-used CI definitions.

- 20.12.2016
  Revised for TSI 1.6 [R6]. Added UCD-1 related CI definitions.

- 12.1.2017
  Revised for software bundle release 1.0.3. Added HDCP Debugging CI definitions, revised UCD-1 and UCD-2 related configuration item definitions.

- 1.3.2017
  Revised for software bundle release 1.0.5. Added DP Sink Link status and DP Sink Link Configuration CI's. Added DisplayPort Reference Source Simple Link test, and related CI's. Added missing CI definition of TSI_HPD_LENGTH.

- 28.4.2017
  Revised for software bundle release 1.1.0. Added USB-Type C related configuration item definitions from the UCD-340 support proposal version 1.1b, section "Configuration items for USB Type-C". Added USB Type-C electrical test definitions, all four "USBC Electrical test set" sections. Added CI definitions relating to USB Type-C electrical tests, section "USB Type-C Electrical tests"

- 10.7.2017
  Revised for TSI 1.8 [R4] and software bundle 1.2.3. Updated electrical test parameter definitions to match with implementation. Added missing USB Type-C electrical test parameter "TSI_USBC_EL_PWR_MEASURE_DELAY". Revised USB Type-C electrical test descriptions (All four "USBC Electrical Test Set" chapters) to match with implementation.

- 14.7.2017
  Fixed mistakes/typos in chapters "USBC Electrical Test Set / DUT as Power Sink" and "USBC Electrical Test Set / DUT as Power Source", Added description for function TSI_DEV_Location. This manual version is still for TSI 1.8 [R4].

- 8.9.2017
  Revised for TSI 1.9 [R2] and software bundle release 1.3.2. Split section for moving the configuration items into section "Configuration item definitions". Added "Generic low-level test results", Re-formatted sections "Reference frames", "Input video format", "Input audio format", "V-by-One inputs", "LVDS Inputs", "Video test" and "Audio test"; Respective old sections were removed. Added new functions for Source functionality, section "Output management functions". Added CI definitions used with Sources, sections "Pattern generator CI definitions" and "Displayport interface specific CI definitions". Removed definitions of unused CI's. Added cross references to RAW results sections from tests that deliver this type of information, All four "USBC Electrical Test Set" Sections.
  Important: *This version of the manual is preliminary.*

- 15.9.2017
  Revised for TSI 1.9 [R3] and software bundle release 1.3.3. Added sections "TSI_R_USBC_IDO_TABLE" and "TSI_USBC_EPU_LOAD_CONTROL".
  Important: *This version of the manual is preliminary.*

- 19.9.2017
  Revised for TSI 1.9 [R4] and software bundle release 1.3.4. Added sections "TSI_USBC_PWR_CONTRACT_CONTROL", "TSI_USBC_PWR_CONTRACT_SELECT", "TSI_USBC_PWR_LOCAL_SINK_PDO", "TSI_USBC_PWR_LOCAL_SOURCE_PDO", "TSI_R_USBC_PWR_REMOTE_SINK_PDO" and "TSI_R_USBC_PWR_REMOTE_SOURCE_PDO".
  Important: *This version of the manual is preliminary.*

- 30.10.2017
  Revised for TSI 1.9 [R6] and software bundle release 1.3.8. Renamed all additional test set related sections to match with actual test names displayed by software applications. Fixed broken cross-references in sections "TSI_VIN_Select", "TSI_TS_SetConfigItem", "TSI_MISC_SetOption" and "Compare video frame sequence with a single reference". Reformatted sections "HDMI Receiver Electrical tests", "DP Receiver electrical tests", "Accessing Info frames", "Miscellaneous", "CRC based video tests" and "DP RefSource simple link test". Respective old sections were removed. Added section "Generic realtime measurements". Revised sections "Miscellaneous", "TSI_W_PG_COMMAND", "TSI_PG_CUSTOM_TIMING_HSYNCW"8, "TSI_PG_CUSTOM_TIMING_VSYNCW", "TSI_PG_CUSTOM_TIMING_FLAGS", "TSI_R_PG_PREDEF_TIMING_COUNT", "TSI_W_PG_PREDEF_TIMING_SELECT" and "Error codes".

- 24.11.2017
  Revised for TSI 1.9 [R7] and software bundle release 1.3.9. Added custom pattern use details to sections "TSI_W_PG_PREDEF_PATTERN_SELECT" and "TSI_PG_CUSTOM_PATTERN_DATA". Added two missing CRC test configuration items "TSI_CRC_MOTION_TEST_ITERATIONS" and "TSI_CRC_COLOR_FORMAT". Revised "TSI_CRC_FRAMES_TO_TEST" to contain correct information. Additional information describing sections "External functions" and "API Base level functions". Some minor cosmetic updates.

- 12.12.2017
  Revised for TSI 1.9 [R8] and software bundle release 1.3.10. Added new USB-Type C parameter "TSI_USBC_EL_MIN_DUT_CURRENT". Revised "USBC Electrical Test Set / DUT as Power Sink".

- 5.1.2018
  Added description of the fourth CRC based video test that was missing from previous version "CRC Based Video Test Set / CRC based continuous sequence of reference frames". Only the manual was updated, no new TSI release was done.

- 16.3.2018
  Revised description of "TSI_DEV_Select" to describe the function more accurately. Revised HDCP debugging "HDCP Debugging configuration items" to also cover Source side.

- 18.5.2018
  Major re-structuring: The old document is now split into two: 1. Original TSI interface, 2. TSI-X Interface (this manual). Removed all cross references from history listing above and replaced with chapter names as re-targeting the cross references is not accurate due to structural changes in the document.

(Continued...)

(...Continued)

- 9.7.2018
  Added TSI_W_SCRI_DELAY, Additional bits to TSI_W_USBC_CABLE_CONTROL and TSI_R_USBC_CABLE_STATUS CI's, Revised typing on pages 13 to 55. Added DP LL CTS Source DUT Tests and Operator feedback support sections.

- 23.8.2018
  Added TSI_R_USBC_PD_STATUS, TSI_W_USBC_PD_COMMAND. Revised DP 1.4 LL CTS test sections.

- 14.12.2018
  Added CI definitions TSI_GLOBAL_LEGACY_DEVICE_HANDLE, TSI_DEVICE_FIRMWARE_INFO, TSI_DEVICE_RELOAD_FW, TSI_DEVICE_HW_RESET. Revised status definitions of CI's TSI_R_HDPC_1X_STATUS and TSI_R_HDCP_2X_STATUS. Revised CRC test ID definitions to have different definitions for HDMI and DP.

- 31.12.2018
  Added CI chapter Memory management and definitions TSI_R_MEMORY_SIZE, TSI_R_MEMORY_MAX_BLOCK_AMOUNT, TSI_W_MEMORY_RESET, TSI_MEMORY_LAYOUT, TSI_MEMORY_BLOCK_INDEX, TSI_W_MEMORY_WRITE, TSI_PG_CUSTOM_PATTERN_INDEX, TSI_PG_CUSTOM_PATTERN_CHAINED_DATA, TSI_PG_CUSTOM_PATTERN_FRAME_TYPE. Extended information about CI TSI_PG_CUSTOM_PATTERN_DATA.

- 16.1.2019
  Added support for MST; DP Sink side CI TSI_R_DPRX_LINK_STATUS_FLAGS updated; Added new CI TSI_DPRX_STREAM_SELECT. Revised CI's TSI_W_DPTX_COMMAND, TSI_PG_ENABLED_STREAM_COUNT. Added sections "Extended Scripting Engine" and "TSI-X TestStand integration". Missing flags added to CI's TSI_DP14_SRCCTS_DUT_CAPS and TSI_DP14_SRCCTS_DUT_TA.

- 5.4.2019
  Revised scripting section of the TSI Reference manual in general. Added MSA support CI's. Added audio parameter definitions for LL CTS tests. Added more detailed control CI's for USB-C Power Objects (for both RDOs and PDOs).

1.2 Acronyms and abbreviations

API
: Application Programming Interface.

UAPI
: Unified Application Programming Interface.

DLL
: Dynamic Link Library.

CI
: Configuration Item.

GUI
: Graphical User Interface.

CTS
: Compliancy Testing System.

MSA
: Main Stream Attributes.

CRC
: Cyclic Redundancy Check.

IDE
: Integrated Development Environment

OS
: Operating System

EDID
: Extended Display Identification Data.

TSI
: Test System Interface

PDO
: Power Delivery Object.

2 INTRODUCTION TO EXTENDED TSI

The Extended TSI API introduces a set of new functions that enable use of multiple physical devices simultaneously from a single process by adding a device handle parameter to the function calls. This new collection of functions is referred to as TSI-X Interface.

TSI still remains fully backwards compatible: The compatibility is achieved by keeping all of the previous functions available in the TSI. The “old” functionality is referred to as Original TSI Interface.

The Original TSI Interface also gains a configuration item that can be used to access and switch between multiple devices. It is important to realize that while it is possible to use multiple devices this way, it does not enable concurrent operations. The downside is that handling multiple devices this way is clumsy and inefficient – multiple threads cannot use multiple devices concurrently through the Original TSI Interface. The upside is that in some cases the second device control needs are typically “Setup & forget”, which can be done through this mechanism more easily.

2.1 TSI Library

TSI Library is currently available for the Windows operating systems. Currently supported OS versions are Windows 7 and later versions of Windows. Windows Server OS editions are not supported.

2.1.1 Features

- Backward compatibility guaranteed: New versions of TSI are guaranteed to be support all functionality of all previous TSI API versions. This means that applications built using TSI SDK can use TSI.DLL published at a later date.

  Important: *Engineering builds and beta releases may introduce features that may not be available and/or may be modified in the actual release. All TSI library copies that are provided outside of the Unigraf Software Bundle and/or as hot-fixes are to be considered as Engineering builds.*

  Important: *The backwards compatibility started from the first official production version of TSI, which was version 1.2.*

- Supports most current Unigraf hardware. Version 2.0 [R0.1] supprots UCD-301, UCD-323, UCD-340 and UCD-400. Additional hardware support will be added.
- All tests are configurable.
- HDCP video and audio testing.
- Electrical testing for production lines to ensure manufacturing quality.
- HTML and text based test logging and reporting support.

2.1.2 Using the TSI API

TSI is a high-level API that is built on top of other API's and/or device drivers that provide low-level access to the supported devices.

Accessing TSI functions

TSI has two function sets to choose from. Each function set has its own loader. To use the "Original TSI" function set, use the loader implementation from "*TSI.C*" and "*TSI.h*". To use the "TSI-X" function set, use the loader implementation from "*TSIX.CPP*" and "*TSIX.H*".

TSI Function return values

All functions in TSI return either TSI_RESULT, or TSI_HANDLE type value. Any function that returns *TSI_RESULT* will do so according to the following rules:

- Negative return value always indicates an error code.
- Zero indicates success. If this value has a specific meaning it is documented in this manual.
- Non-zero positive values indicate success and additionally convey information. This manual will define the meaning of these values if they are known at the time of writing this manual: Do not assume that a function returning zero today will return zero also in the future versions of TSI.

Application should check the return value of every TSI function call. Please make sure to check the return values properly.

- Checking for generic error: If the function return value is less than *TSI_SUCCCESS*, the function has failed.

```
if(Result < TSI_SUCCESS) { /* HANDLE ERROR */ }
```

- Checking for generic success: If the function return value is equal or greater than *TSI_SUCCESS*, the function has succeeded.

```
if(Result >= TSI_SUCCESS) { /* SUCCESS */ }
```

Any function returning *TSI_HANDLE* type will do so according to the following rules:

- If the function failed, the return value is NULL. The NULL value is considered as invalid handle.
- If the function succeeded, the return value is non-NULL.

Application should check returned handle values before attempting to use them with consequent function calls.

- Checking for generic error: If the returned handle value is NULL, the function has failed.

```
if(Handle == NULL) { /* HANDLE ERROR */ }
```

- Checking for generic success: If the returned handle value is not NULL, the function has succeeded.

```
if(Handle != NULL) { /* SUCCESS */ }
```

Important: *The return values defined per function do not override these rules, unless specifically indicated that the rules are violated.*

Thread safety

TSI API is protected against harmful concurrent access. For the Original TSI, this protection prevents concurrent use of the internal handle value. For the TSI-X the protection works on device level, preventing concurrent operations on the device, unless the operations are specifically enabled.

Firmware Versions

TSI does not communicate directly with hardware, instead it uses lower level APIs to do so. As a result, TSI has no specific requirements of firmware versions.

Low-level API versions

TSI is provided as part of a bundle of software. The software bundle includes the low-level API libraries. In general, TSI expects to use the version provided in the bundle with the TSI library itself – however, this is not a strict requirement. TSI only requires that the low-level API is equal or later than a specified minimum version.

2.1.3 Windows systems

The TSI Library is provided as DLL files, named "TSI.DLL". It is available as 32-bit (x86) edition and also as 64-bit (x64) edition. The 32-bit edition can be used in any version of Windows edition that supports 32-bit editions. The 64-bit edition can only be used in 64-bit editions of Windows. Please see below installation details for each type of system.

Windows x86 editions

The software bundle installer will optionally install the 32-bit edition of TSI.DLL into the operating systems shared program-files location. The actual folder is determined during installation procedure and may be different depending on OS installation options and language. For English language, this folder is typically "*C:\Program files\Common Files\Unigraf\Shared*". The folder is accessible by entering ***cd "%programfiles%"\Common Files\Unigraf\Shared*** on the command line.

Windows x64 editions

The software bundle installer will optionally install the 32-bit edition of TSI.DLL into the operating systems shared 32-bit program-files location. The actual folder is determined during installation procedure and may be different depending on OS installation options and language. For English language, this folder is typically "*C:\Program files (x86)\Common files\Unigraf\Shared*". Additionally, the 64-bit edition of TSI.DLL is similarly installed into the shared 64-bit program-files location. For English language installations, this folder is typically "*C:\Program Files \Common Files\Unigraf\Shared*". You can access the 64-bit folder by entering ***cd "%programfiles%"\Common files\unigraf\shared*** on the command line.

Windows – All editions

Starting with Software Bundle series 1.5, the optional TSI SDK is installed into the 'all users' profile directory to avoid usage problems with UAC. The location of this directory is determined during installation procedure and may be different depending on OS installation options and language. For English language this folder is typically "*C:\Users\All Users\Unigraf\TSI_SDK*". You should be able to access the installation folder on the command-line by entering ***cd "%programdata%"\Unigraf\TSI_SDK***.

2.2 Original TSI and TSI-X compared

There are a number of differences and similarities between the two interfaces. This section focuses on describing the differences as well as the similarities between the two interfaces.

2.2.1 Function name comparison

There are a number of functions that are identical in both TSI-X and the Original TSI interfaces. These functions are listed below:

- TSI_Init
- TSI_Clean
- TSI_DEV_GetDeviceCount
- TSI_DEV_GetDeviceInfo
- TSI_DEV_GetDeviceName
- TSI_DEV_GetParameterCount
- TSI_DEV_GetParameterID
- TSI_DEV_SetSearchMask
- TSI_APREV_GetDeviceCount
- TSI_APREV_GetDeviceName
- TSI_MISC_GetErrorDescription

(Continued…)

(...Continued)

The following Original TSI interface functions have a mostly direct counterpart in TSI-X, with the added device handle parameter. The TSI-X side functions use the prefix TSIX. Below is a list of the Original TSI interface functions, and their TSI-X counterparts:

- TSI_VOUT_Disable → TSIX_VOUT_Disable
- TSI_VOUT_Enable → TSIX_VOUT_Enable
- TSI_VOUT_Select → TSIX_VOUT_Select
- TSI_VOUT_GetOutputName → TSIX_VOUT_GetOutputName
- TSI_VOUT_GetOutputCount → TSIX_VOUT_GetOutputCount
- TSI_VOUT_GetParameterID → TSIX_VOUT_GetParameterID
- TSI_VOUT_GetParameterCount → TSIX_VOUT_GetParameterCount
- TSI_VIN_Disable → TSIX_VIN_Disable
- TSI_VIN_Enable → TSIX_VIN_Enable
- TSI_VIN_Select → TSIX_VIN_Select
- TSI_VIN_GetInputName → TSIX_VIN_GetInputName
- TSI_VIN_GetInputCount → TSIX_VIN_GetInputCount
- TSI_VIN_GetParameterID → TSIX_VIN_GetParameterID
- TSI_VIN_GetParameterCount → TSIX_VIN_GetParameterCount
- TSI_STLOG_WaitMessage → TSIX_STLOG_WaitMessage
- TSI_STLOG_GetMessageCount → TSIX_STLOG_GetMessageCount
- TSI_STLOG_GetMessageData → TSIX_STLOG_GetMessageData
- TSI_STLOG_Clear → TSIX_STLOG_Clear
- TSI_MISC_LoadReference → TSIX_MISC_LoadReference
- TSI_MISC_SaveReference → TSIX_MISC_SaveReference
- TSI_MISC_SetOption → TSIX_MISC_SetOption
- TSI_TS_CaptureReference → TSIX_TS_CaptureReference
- TSI_TS_Clear → TSIX_TS_Clear
- TSI_TS_GetConfigItem → TSIX_TS_GetConfigItem
- TSI_TS_SetConfigItem → TSIX_TS_SetConfigItem
- TSI_TS_GetTestCount → TSIX_TS_GetTestCount
- TSI_TS_GetTestInfo → TSIX_TS_GetTestInfo
- TSI_TS_GetTestParameterCount → TSIX_TS_GetTestParameterCount
- TSI_TS_GetTestParameterID → TSIX_TS_GetTestParameterID
- TSI_TS_LoadConfig → TSIX_TS_LoadConfig
- TSI_TS_SaveConfig → TSIX_TS_SaveConfig
- TSI_TS_RunTest → TSIX_TS_RunTest
- TSI_TS_WaitInputSignal → TSIX_TS_WaitInputSignal
- TSI_REP_BeginLogRecord → TSIX_REP_BeginLogRecord
- TSI_REP_EndLogRecord → TSIX_REP_EndLogRecord
- TSI_VPREV_SetWindowHandle → TSIX_VPREV_SetWindowHandle
- TSI_APREV_SetWindowHandle → TSIX_APREV_SetWindowHandle
- TSI_APREV_SelectDevice → TSIX_APREV_SelectDevice

The Original TSI interface function *TSI_DEV_Select(...)* has no direct counterpart in TSI-X; Instead its functioncality is divided between the two new TSI-X interface functions: *TSIX_DEV_OpenDevice(...)* and *TSIX_DEV_CloseDevice(...)*.

2.2.2 Behavioral differences

The biggest behavioral difference between the original TSI and the TSI-X -interface is that the TSI-X does not automatically select default device, input nor output. The client application must do this explicitly when using the TSI-X interface.

The Original TSI interface is protected against concurrent access for most operations. However, the TSI-X interface has concurrent access protection per device only. This allows TSI-X to perform all operations on multiple devices concurrently.

2.2.3 Changes in both interfaces

The new TSI 2.x series library is a re-write of the 1.x series. The goals for this re-write were to introduce access to multiple devices from a single application while retaining backwards compatibility. In addition, one of the side goals was to remove certain aspects of internal operations that were not only wasting system resources, but also slowing down test operations.

One of the most important changes is that the new TSI library does not run video / audio capture on the background, unless it is required by some functionality (such as video or audio preview etc…). As a result, the new library is typically more responsive and uses less system resources when idle.

Configuration item scope is changed so that almost all configuration items are now associated with a specific device instead of being globally available in general. In addition to changing the previous global scope to device scope, a new global scope is also available. The global scope CI's are different from the others as they can be accessed with any device handle (including invalid ones). Typically, the global scope CI's concern the entire TSI library.

2.3 Selecting the TSI interface to use

As there are now two function sets, it obviously introduces an additional complexity to software developers using TSI. The following points can hopefully help developers to make the right choices for their projects.

Interface selection for new applications

When starting a new application development, the question of "which TSI interface to use" is the most complicated one, as both interfaces are viable options. So, the selection needs to take into account a developer's previous experience with TSI, as well as project requirements and how likely the requirements are to grow.

Select TSI-X Interface if:

1. Application developer(s) are not familiar with TSI development.

   - OR-

2. Project requires use of multiple Unigraf devices from a single application.

   - OR-

3. Project can be expected to grow bigger over time and/or the resulting software is going to be in use for long time.

Select Original TSI Interface if:

1. Application developer(s) are already familiar with TSI development.

   - AND -

2. The project in question does not require use of multiple devices.

Interface selection for existing applications

If the existing application does not require use of multiple Unigraf devices at a time, there is no need to migrate to TSI-X.

If the existing application needs to control two Unigraf devices: One is used as source, and one is used as sink, with DUT in between. Only one of the Unigraf devices is used as TE at a time, while the other is just providing basic sink/source functionality. In these cases, one of the devices perform most of the operations while the other device sees very little action and the operations do not need to overlap – and so it is probably better to stay with the Oringinal TSI Interface. (As the time spent on migrating to the new TSI-X probably doesn't justify the benefits, which are relatively small in this case).

If the existing application needs to use more than one Unigraf devices concurrently, then it is recommended to migrate to TSI-X. Please note that recommendation does not equal requirement: It is still possible to use the Original TSI interface. However, in this scenario, TSI-X can offer significantly better performance with less complex application.

(Continued...)

(...Continued)

Enabling the TSI-X interface in your application

The TSI-X interface is exposed by the TSI loader named “TSIX”. Include “TSIX.cpp” and “TSIX.h” to your project, and call the TSIX_LoadAPI function. After this, the TSI-X interface functions are available for use.

Important: *The Original TSI and TSI-X interface loaders are designed to be mutually exclusive. Only select one or the other for your program. Do not try using both.*

Enabling the Original TSI interface in your application

The Original TSI interface is exposed by the TSI loader named “TSI”. Include “TSI.c” and “TSI.h” to your project, and call the TSI_LoadAPI(…) function. After this the, original TSI interface functions are available for use.

Important: *The Original TSI and TSI-X interface loaders are designed to be mutually exclusive. Only select one or the other for your program. Do not try using both*

2.4 Migrating from Original TSI interface to TSI-X

While the migration to the new interface might seem necessary, it is also important to realize that it is not required for all applications. Before starting the migration, make sure you have read Section “*2.3 Selecting the TSI interface to use*”, and concluded that migration is in your best interests.

The major difference in operations between the Original TSI interface and TSI-X interface is the automatic device selection: It does not exist in TSI-X. Therefore, Original TSI interface applications may have skipped explicit device/input and/or output selections. In TSI-X these steps can’t be skipped anymore. This means that after loading the TSI.DLL, there must be explicit calls to TSI_Init, TSIX_DEV_OpenDevice and (depending on what the application is doing) a call to either TSIX_VIN_Select + TSIX_VIN_Enable and/or TSIX_VOUT_Select + TSIX_VOUT_Enable.

To start the migration procedure, create a backup of your entire project as a safeguard in case of a migration failure.

A good logical place to start making changes to the program code is to switch the TSI interface to TSI-X. Remove “TSI.C” and “TSI.H” from your project, and add “TSIX.CPP” and “TSIX.H”. Replace “TSI.H” includes with “TSIX.h”. After this, replace the TSI_DEV_Select function with a call to TSIX_DEV_OpenDevice, and store the returned handle. This parameter is later used with almost all TSIX functions to identify the device to interact with. After this change, proceed to rename existing TSI functions (except for the identical ones). See the table in Section *2.2.1 Function name comparison* for functions that are identical, and functions that need to be renamed. When renaming the functions, remember to add the device handle parameter.

Finally, check the code at logical level: the Original TSI applications required fewer steps, when compared with the TSI-X applications. If a function is returning errors which were not seen previously with the original TSI, chances are that a required function call is missing before the function.

3 FUNCTIONS

This section describes the TSI client callable functions.

3.1 External functions

The functions in this chapter are delivered as source code in files “TSI.C” and “TSI.H”. Please include these files to your project to access TSI_LoadAPI and TSI_UnloadAPI functions.

3.1.1 TSI_LoadAPI

```
TSI_RESULT __stdcall TSI_LoadAPI
(
     _TCHAR *LibName
);
```

Synopsis

Load the API DLL, and resolve all the service functions. After this call, the API client functions are available for use. If the DLL load is successful, the function continues to call the API Init function on behalf of the client application. When calling the Init function, the LoadAPI function will use the client version constant in TSI_Types.h.

Important: *If the LibName parameter is NULL, the loader will look for TSI.DLL from the default install locations first, and from OS search paths subsequently. If you wish to have TSI.DLL loaded from OS search paths only, you should give pointer to string containing “TSI.DLL” as parameter to this function.*

Parameters

LibName

A NULL terminated string containing the name (and, optionally, path) of TSI.DLL. This parameter can be NULL: If the parameter is NULL, then the default install location is attempted first, followed by system default search paths.

Result

If the function succeeds, the return value is a non-zero positive number indicating the number of times the Init function has been called.

Important: *If the DLL was loaded successfully, but the TSI_Init() function failed, the return value is zero. In this case it will be necessary to call the TSI_Init() function again.*

If the function fails, the return value is a negative error code.

See Also

3.1.2 TSI_UnloadAPI, 3.2.1 TSI_Init

3.1.2 TSI_UnloadAPI

```
TSI_RESULT __stdcall TSI_UnloadAPI();
```

Synopsis

This function will first call the TSI_Clean( ) function of the API. If the TSI_Clean( ) return value is zero, this function continues to free the TSI.DLL and release API loader resources.

Important: *If the TSI_Clean( ) call was not called for last time, the function will call TSI_Init( ) and return with an error status.*

Important: *Receiving an error result from this function indicates a resource management issue in the application.*

Result

If the function succeeds, the return value is zero. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, the return value is a negative error code.

See Also

3.2.2 TSI_Clean, 3.2.1 TSI_Init, 3.1.1 TSI_LoadAPI

3.2 TSI System functions

3.2.1 TSI_Init

ClientVersion 1, and higher	*No license requirements*

```
TSI_RESULT __stdcall TSI_Init
(
    TSI_VERSION_ID ClientVersion
);
```

Synopsis

Initializes the API for use and sets up the internal compatibility flags based on the given *ClientVersion* value. Calls to *TSI_Init( )* are reference counted: *Clean( )* must be called equal number of times for correct operation. If *TSI_Init( )* is not called, all service functions will fail with *TSI_ERROR_NOT_INITIALIZED*.

Parameters

ClientVersion

Indicates the TSI_Types.h file's version used to call the API functions. Always use the *TSI_CURRENT_VERSION* define as parameter when calling *TSI_Init( )* to ensure compatibility with later versions of the DLL.

Important: *The first call to* TSI_Init( ) *will set the compatibility layer for the entire process. Following calls are required to use same* ClientVersion *value. If the* ClientVersion *is different between two calls, the later function-call will fail with* TSI_ERROR_COMPATIBILITY_MISMATCH.

Important: *If the requested* ClientVersion *is NOT supported by the loaded DLL, then this function will fail with* TSI_ERROR_NOT_COMPATIBLE.

Result

If the function succeeds, the return value is a non-zero positive value indicating the API reference count after the function call.

If the function fails, the return value is a negative error code.

See Also

3.2.2 TSI_Clean

3.2.2 TSI_Clean

ClientVersion 1, and higher *No license requirements*

```
TSI_RESULT __stdcall TSI_Clean();
```

Synopsis

Closes device and releases API resources if the reference count after the function call equals zero. Calls to *TSI_Init( )* are reference counted: *Clean( )* must be called equal number of times for correct operation.

Result

If the function succeeds, the return value is a positive value (or zero) indicating the API reference count after the function call. If the return value is zero, the API functions are not available after this function call.

See Also

3.2.1 TSI_Init

3.3 Device management functions

3.3.1 TSI_DEV_GetParameterCount

ClientVersion 4, and higher *No license requirements*

```
TSI_RESULT __stdcall TSI_DEV_GetParameterCount
(
    TSI_DEVICE_ID DeviceID
);
```

Synopsis

Retrieves the number of parameters that can change a device's behavior when the device is selected. To read the list of parameters, please iterate through it by calling the *TSI_DEV_GetParameterID* function in a loop.

Important: *Use of this function is not needed for applications that only use known device types, or applications that always expect default behavior on device selection.*

Parameters

DeviceID

Identifies the device from which to read the parameter count. Valid *DeviceID* values range from zero to the number of devices returned by *TSI_DEV_GetDeviceCount* minus one.

Result

If the function succeeds, the return value is a positive value indicating the number of configuration items that can change the device's behavior during device selection.

If the return value is zero, there are no configuration items that could change the device's behavior when it is selected.

If the function fails, the return value is a negative error code.

See Also

3.3.2 TSI_DEV_GetParameterID,3.3.8 TSIX_DEV_OpenDevice, 3.3.5 TSI_DEV_GetDeviceCount

3.3.2 TSI_DEV_GetParameterID

ClientVersion 4, and higher *No license requirements*

```
TSI_RESULT __stdcall TSI_DEV_GetParameterID
(
    TSI_DEVICE_ID DeviceID,
    int ParameterIndex,
    TSI_CONFIG_ID *ParamID,
    unsigned int *ParamFlags
);
```

Synopsis

Retrieves information about a configuration item that may effect the behavior of a device while it's being selected. Some devices may have features that must be enabled or configured during device opening. This function exists to dynamically resolve any such configuration items per device.

Important: *Use of this function is not needed for applications that only use known device types, or applications that always expect default behavior on device selection.*

Parameters

DeviceID

Identifies the device from which to get the configuration ID value. Valid *DeviceID* values range from zero (0) to the number of devices returned by *TSI_DEV_GetDeviceCount* minus one.

ParameterIndex

Identifies the index of the parameter being queried. The first valid index is zero (0). Last valid index is value returned by successful call to *TSI_DEV_GetParameterCount* minus one.

ParamID

Pointer to *TSI_CONFIG_ID* type variable, which will receive a configuration item ID value.

ParamFlags

Pointer to an unsigned int type variable, which will receive configuration item related flag information.

Result

If the function succeeds, the return value is zero and information about a configuration item is placed to variables pointed by *ParamID* and *ParamFlags*. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, the return value is a negative error code. The variable contents pointed by *ParamID* and *ParamFlags* remain unchanged.

See Also

3.3.1 TSI_DEV_GetParameterCount, 3.3.5 TSI_DEV_GetDeviceCount

3.3.3 TSI_DEV_SetSearchMask

ClientVersion 3, and higher *No license requirements*

```
TSI_RESULT __stdcall TSI_DEV_SetSearchMask
(
    TSI_DEVICE_CAPS RequiredCaps,
    TSI_DEVICE_CAPS UnallowedCaps
);
```

Synopsis

Limit number of devices found. Not all TSI applications support all features of TSI, and as such it may be necessary to limit the number of devices listed in the device list. For example, an application such as AV Test does not need to find devices that don't have support for video capture.

Important: *Versions 1 and 2 clients will have RequiredCaps and UnallowedCaps pre-defined so that the operation remains identical.*

Important: *Version 3 and later clients will default to listing all devices present.*

Parameters

RequiredCaps

Flag bits that define which features are required for listed devices.

Important: *Do not issue capability bits that are undefined. If an undefined capability bit is set, the function will fail.*

UnallowedCaps

Flag bits that define which features must not be present on listed devices.

Important: *Do not issue capability bits that are undefined. If an undefined capability bit is set, the function will fail.*

Result

If the function succeeds, the return value is a positive value indicating the number of supported capture device attached to the local system. If there are no supported device present, the return value is zero.

If the function fails, the return value is a negative error code.

See Also

-

3.3.4 TSI_DEV_GetDeviceInfo

ClientVersion 3, and higher *No license requirements*

```
TSI_RESULT __stdcall TSI_DEV_GetDeviceInfo
(
    TSI_DEVICE_ID DeviceID,
    TSI_DEVICE_CAPS *Caps
);
```

Synopsis

Retrieves device capabilities bit-field of the indicated device.

Parameters

DeviceID

Identifies device from which to retrieve the capabilities flags. Valid *DeviceID* values range from zero to the number of devices returned by *TSI_DEV_GetDeviceCount* minus one.

Caps

Pointer to *TSI_DEVICE_CAPS* bit-field, which will receive the capabilities flags.

Result

If the function succeeds, the return value is zero. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, the return value is a negative error code.

See Also

3.3.5 TSI_DEV_GetDeviceCount

3.3.5 TSI_DEV_GetDeviceCount

ClientVersion 1, and higher *No license requirements*

```
TSI_RESULT __stdcall TSI_DEV_GetDeviceCount();
```

Synopsis

Enumerates supported Unigraf devices attached to the local system and returns the number of devices found. Please note that in some cases a hardware device may not directly map into a single TSI device: Depending on the device itself, and its low-level features, one hardware device may appear many times in TSI device enumeration. One example of such devices are the UCD family of devices. To form device ID used with other functions which require reference to a device use a number starting from zero (0) to the number returned by this function minus one.

Result

If the function succeeds, the return value is a positive value indicating the number of supported capture device attached to the local system. If there are no supported devices present, the return value is zero.

If the function fails, the return value is a negative error code.

See Also

3.3.6 TSI_DEV_GetDeviceName, 3.3.8 TSIX_DEV_OpenDevice

3.3.6 TSI_DEV_GetDeviceName

ClientVersion 1, and higher *No license requirements*

```
TSI_RESULT __stdcall TSI_DEV_GetDeviceName
(
    TSI_DEVICE_ID_ID DeviceID,
    char *DevNameString,
    unsigned int NameStringMaxLength
);
```

Synopsis

Retrieves a human readable name to identify the device associated to a DeviceID.

Parameters

DeviceID

A numeric value identifying the queried device. Valid *DeviceID* values range from zero to the number of devices returned by *TSI_DEV_GetDeviceCount* minus one.

DevNameString

Pointer to an array of characters that will receive the device's name. The string is guaranteed to be NULL terminated. If the buffer is not large enough to store the full name, the string is truncated.

NameStringMaxLength

Length of the DevNameString character array in chars. The recommended buffer size is 64 chars or more.

Result

If the function succeeds, the return value is the number of chars required by the full name of the device regardless of the *NameStringMaxLength* parameter. If the returned value is EQUAL or HIGHER than *NameStringMaxLength*, it means that the name was truncated.

If the function fails, the return value is a negative error code.

See Also

3.3.5 TSI_DEV_GetDeviceCount, 3.3.5 TSI_DEV_GetDeviceCount

3.3.7 TSI_DEV_Location

ClientVersion 11, and higher *No License requirements*

```
TSI_RESULT __stdcall TSI_DEV_Location
(
    const char * Location
);
```

Synopsis

Provides hints about the realm of device discovery used by *TSI_DEV_GetDeviceCount*.

Important: *This function is not needed if all devices are connected via USB.*

Parameters

Location

A string which is a combination of location hints. Each hint is represented as key-value pair. Hints are separated with a ';' sign. Hints are of the following types:

- network=x.x.x.x

  This specifies sub-network mask for discovery broadcast, e.g. 192.168.1.255

Result

If the function succeeds, the return value is zero. Please notice that future versions may return positive values indicating success.

If the function fails, the return value is a negative error code.

3.3.8 TSIX_DEV_OpenDevice

ClientVersion 12, and higher *No license requirements*

```
TSI_HANDLE __stdcall TSIX_DEV_OpenDevice
(
    TSI_DEVICE_ID DeviceID,
    TSI_RESULT *Result;
);
```

Synopsis

Open the indicated device and return a handle to identify the device for other functions. Unlike the *DeviceID* values which can change dynamically, handle values remain constant until the device is closed. If the device can't be opened, the handle value will be NULL. If the *TSI_RESULT* variable pointer is non-NULL, the result variable is also set accordingly.

Important: *While the DeviceID may identify a logical device, it is the physical device that is opened. This means that if a physical device is shown as multiple virtual devices, only one of these virtual devices can be accessed.*

Parameters

DeviceID

Identifies the device to open from the current list of devices.

Result

Pointer to a *TSI_RESULT* type variable. The pointer value can be NULL. If the pointer's value is NULL, the result code is not set. The variable will be set according to the result of the function. If the open succeeds, the *Result* variable will be set to zero, or non-zero positive value. If the open fails, the *Result* variable will be set to a negative value indicating the appropriate error code.

Result

If the function succeeds, the return value is a non-NULL handle value identifying the opened device. If set, the *Result* value is also set to zero, or positive value indicating success.

If the function fails, the return value is NULL. If set, the *Result* variable's value is set to a negative value indicating an error code.

See Also

3.3.9 TSIX_DEV_CloseDevice

3.3.9 TSIX_DEV_CloseDevice

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSI_DEV_CloseDevice
(
    TSI_HANDLE Device
);
```

Synopsis

Close the device indicated by the handle value and release all related resources; For example, re-opening the device will not restore any of its settings. If the device is closed successfully, the handle value becomes invalid.

Parameters

Device

Handle indicating the device to close. This handle is received from *3.3.8 TSIX_DEV_OpenDevice* function.

Result

If the function succeeds, the return value is zero or a positive value.

If the function fails, the return value is negative indicating an error code.

See Also

3.3.8 TSIX_DEV_OpenDevice

3.4 Input management functions

3.4.1 TSIX_VIN_GetParameterCount

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_VIN_GetParameterCount
(
    TSI_HANDLE Device,
    TSI_INPUT_ID InputID
);
```

Synopsis

Retrieves the number of parameters that changes an input's behavior when the input is being selected. To read the list of parameters, iterate through it by calling *TSIX_VIN_GetParameterID* in a loop.

Important: *Use of this function is not needed for applications that only use known device types, or applications that always expect default behavior on input selection.*

Parameters

Device

Indicates the device on which the operation is to be carried out.

InputID

Identifies the input from which to read the parameter count. Valid *InputID* values range from zero (0) to the number of inputs returned by *TSIX_VIN_GetInputCount* function minus one.

Result

If the function succeeds, the return value is a positive value indicating the number of configuration items that changes the input's behavior during input selection.

If the return value is zero, there are no configuration items that could change the input's behavior when it is selected.

If the function fails, the return value is a negative error code.

See Also

3.4.2 TSIX_VIN_GetParameterID, 3.4.3 TSIX_VIN_GetInputCount

3.4.2 TSIX_VIN_GetParameterID

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_VIN_GetParameterID
(
    TSI_HANDLE Device,
    TSI_INPUT_ID InputID,
    int ParameterIndex,
    TSI_CONFIG_ID *ParamID,
    unsigned int *ParamFlags
);
```

Synopsis

Retrieves information about a configuration item that changes the behavior of an input while it is being selected. Some inputs may have features that must be enabled or configured during input activation. The function exists to dynamically resolve any such configuration items per input.

Important: *Use of this function is not needed for applications that only use known device types, or applications that always expect default behavior on input selection.*

Parameters

Device

Indicates the device on which the operation is to be carried out.

InputID

Identifies the input from which to get the configuration ID value. Valid *InputID* values range from zero (0) to the number of inputs returned by *TSIX_VIN_GetInputCount* function minus one.

ParameterIndex

Identifies the index of the parameter being queried. The first valid index is zero (0). Last valid index is value returned by successful call to *TSIX_VIN_GetParameterCount* minus one.

ParamID

Pointer to *TSI_CONFIG_ID* type variable, which will receive a configuration item ID value.

ParamFlags

Pointer to an unsigned int type variable, which will receive configuration item related flag information.

(Continued…)

(...Continued)

Result

If the function succeeds, the return value is zero and information about a configuration item is placed to variables pointed by *ParamID* and *ParamFlags*. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, the return value is a negative error code. The variable contents pointed by *ParamID* and *ParamFlags* remain unchanged.

See Also

3.4.1 TSIX_VIN_GetParameterCount, 3.4.3 TSIX_VIN_GetInputCount, 3.4.5 TSIX_VIN_Select

3.4.3 TSIX_VIN_GetInputCount

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_VIN_GetInputCount
(
    TSI_HANDLE Device
);
```

Synopsis

Returns the number of inputs on the active capture device. Input ID Values range from zero (0) to the value returned by this function.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Result

If the function succeeds, the return value is a non-zero positive value indicating the number of audio/video interfaces present on the active device.

If the function fails, the return value is a negative error code.

See Also

3.4.4 TSIX_VIN_GetInputName, 3.4.5 TSIX_VIN_Select

3.4.4 TSIX_VIN_GetInputName

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_VIN_GetInputName
(
    TSI_HANDLE Device,
    TSI_INPUT_ID InputID,
    char *InputNameString,
    unsigned int NameStringMaxLen
);
```

Synopsis

Retrieve a human readable name for the input associated with the given *InputID*.

Parameters

Device

Indicates the device on which the operation is to be carried out.

InputID

ID value of the input to be identified. Valid *InputID* values range from zero (0) to the number of inputs returned by *TSIX_VIN_GetInputCount* function minus one.

InputNameString

Pointer to an array of characters that will receive a human readable name of the input. The resulting string is guaranteed to be NULL terminated. If the available string space is not long enough to contain the full name, the string is truncated.

NameStringMaxLen

Number of characters available in the InputNameString buffer. The recommended size of the Input name is 64 characters, or more.

Result

If the function succeeds, the return value is the number of characters required by the full input name regardless of *NameStringMaxLen* parameter. If the returned value is EQUAL or HIGHER than *NameStringMaxLen*, it means that the name string was truncated.

If the function fails, the return value is a negative error code.

See Also

3.4.3 TSIX_VIN_GetInputCount, 3.4.3 TSIX_VIN_GetInputCount

3.4.5 TSIX_VIN_Select

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_VIN_Select
(
    TSI_HANDLE Device,
    TSI_INPUT_ID InputID
);
```

Synopsis

Selects an audio/video input to be activated. The intention of this function is to provide client application means to select one input out of many.

Parameters

Device

Indicates the device on which the operation is to be carried out.

InputID

Identifies the input to be activated. Valid *InputID* values range from zero (0) to the number of inputs returned by *TSIX_VIN_GetInputCount* function minus one.

Result

If the function succeeds, the return value is zero, or non-zero positive value.

If the function fails, the return value is a negative error code.

See Also

3.4.3 TSIX_VIN_GetInputCount, 3.4.6 TSIX_VIN_Enable

3.4.6 TSIX_VIN_Enable

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_VIN_Enable
(
    TSI_HANDLE Device,
    TSI_FLAGS Flags
);
```

Synopsis

Enable audio/video input. This function can succeed only if the input status at time of call is disabled. When an input becomes enabled, TSI may do some initialization of it's features. The exact initialization depends on the device type, and on the type of the input.

The input's enable state works as a gate-keeper for operations on that input. If the input is disabled, TSI will not alter the state of the input.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Flags

Obsolete. Any value passed here is ignored.

Result

If the function succeeds, the return values is zero. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, the return value is a negative error code.

See Also

3.3.8 TSIX_DEV_OpenDevice, 3.4.5 TSIX_VIN_Select, 3.4.7 TSIX_VIN_Disable

3.4.7 TSIX_VIN_Disable

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_VIN_Disable
(
    TSI_HANDLE Device
);
```

Synopsis

Disables the audio/video input. When the input is disabled, its state can't be modified.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Result

If the function succeeds, the audio/video input state is disabled and the functions return value is zero. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, the return value is a negative error code.

See Also

3.4.6 TSIX_VIN_Enable

3.5 Output management functions

3.5.1 TSIX_VOUT_GetParameterCount

ClientVersion 12, and higher *<license:TBD>*

```
TSI_RESULT __stdcall TSIX_VOUT_GetParameterCount
(
    TSI_HANDLE Device,
    TSI_OUTPUT_ID OutputID
);
```

Synopsis

Retrieves the number of parameters that changes an output's behavior when output is being selected. To read the list of parameters, iterate through it by calling *TSI_VOUT_GetParameterID* in a loop.

Important: *Use of this function is not needed for applications that use known device types, or applications that always expect default behavior on output selections.*

Parameters

Device

Indicates the device on which the operation is to be carried out.

OutputID

Identifies the output from which to read the parameter count. Valid *OutputID* values ranges from zero (0) to the number of outputs returned by *TSI_VOUT_GetOutputCount* function minus one.

Result

If the function succeeds, the return value is a positive value indicating the number of configuration items that changes the output's behavior during output selection.

If the return value is zero, there are no configuration items that could change the output's behavior when it is selected.

If the function fails, the return value is a negative error code.

See Also

-

3.5.2 TSIX_VOUT_GetParameterID

ClientVersion 12, and higher *<license:TBD>*

```
TSI_RESULT __stdcall TSIX_VOUT_GetParameterID
(
    TSI_HANDLE Device,
    TSI_OUTPUT_ID OutputID,
    int ParameterIndex,
    TSI_CONFIG_ID *ParamID
    unsigned int *ParamFlags
);
```

Synopsis

Retrieves information about a configuration item that changes the behavior of an output while it is being selected. Some outputs may have features that must be enabled or configured during output activation. The function exists to dynamically resolve any such configuration items per output.

Important: *Use of this function is not needed for applications that use known device types, or applications that always expect default behavior on output selections.*

Parameters

Device

Indicates the device on which the operation is to be carried out.

OutputID

Identifies the output from which to get the configuration ID value. Valid *OutputID* values range from zero (0) to the number of outputs returned *TSI_VOUT_GetOutputCount* function minus one.

ParameterIndex

Identifies the index of the parameter being queried. The first valid index is zero (0). Last valid index is value returned by successful call to *TSI_VOUT_GetParameterCount* minus one.

ParamID

Pointer to *TSI_CONFIG_ID* type variable, which will receive a configuration item ID value.

ParamFlags

Pointer to an unsigned int type variable, which will receive configuration item related flag information.

(Continued...)

(...Continued)

Result

If the function succeeds, the return value is zero and information about a configuration item is placed to variables pointed by *ParamID* and *ParamFlags*. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, the return value is a negative error code. The variable contents pointed by *ParamID* and *ParamFlags* remain unchanged.

See Also

-

3.5.3 TSIX_VOUT_GetOutputCount

ClientVersion 12, and higher *<license:TBD>*

```
TSI_RESULT __stdcall TSIX_VOUT_GetOutputCount
(
    TSI_HANDLE Device
);
```

Synopsis

Returns the number of outputs on active device. Output ID values range from zero (0) to the value returned by this function (minus one). If no device is active, this function will activate the device that has device ID of zero.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Result

If the function succeeds, the return value is a non-zero positive value indicating the number of output interfaces present on the active device.

If the return value is zero, it means there are no output interfaces present on the active device.

If the function fails, the return value is a negative error code.

See Also

-

3.5.4 TSIX_VOUT_GetOutputName

ClientVersion 12, and higher *<license:TBD>*

```
TSI_RESULT __stdcall TSIX_VOUT_GetOutputName
(
     TSI_HANDLE Device,
     TSI_OUTPUT_ID OutputID,
     char *OutputNameString,
     unsigned int NameStringMaxLength
);
```

Synopsis

Retrieves a human readable name for the output associated with the given *OutputID*.

Parameters

Device

Indicates the device on which the operation is to be carried out.

OutputID

ID value of the output to be identified. Valid *OutputID* values range from zero (0) to the number of outputs returned by *TSI_VOUT_GetOutputCount* function minus one.

OutputNameString

Pointer to an array of characters that will receive a human readable name of the output. The resulting string is guaranteed to be NULL terminated. If the available string space is not long enough to contain the full name, the string is truncated.

NameStringMaxLength

Number of characters available in the OutputNameString buffer. The recommended length of the output name is 64 characters, or more.

Result

If the function succeeds, the return value is the number of characters required by the full output name regardless of *NameStringMaxLen* parameter. If the return value is EQUAL or HIGHER than *NameStringMaxLen*, it means that the name string was truncated.

If the function fails, the return value is zero.

See Also

-

3.5.5 TSIX_VOUT_Select

ClientVersion 12, and higher *<license:TBD>*

```
TSI_RESULT __stdcall TSIX_VOUT_Select
(
    TSI_HANDLE Device,
    TSI_OUTPUT_ID OutputID
);
```

Synopsis

Selects an audio/video output for use. The intention of this function is to provide client application means to select one output out of many.

Important: *Operation changed in* ClientVersion *12! Starting with* ClientVersion *12, the selected output is not enabled automatically. This changed was done so that it would work the same as the* TSIX_VIN_Select.

Important: *If an application declared* ClientVersion *11 (or lower) when the* TSI_Init( ) *was called, the output is still automatically enabled in order to keep the backwards compatibility!*

Parameters

Device

Indicates the device on which the operation is to be carried out.

OutputID

Identifies the output to be configurable. Valid *outputID* values range from zero (0) to the number of inputs returned by *TSI_VOUT_GetOutputCount* functions minus one.

Result

If the function succeeds, the return value is zero.

If the function fails, the return value is a negative error code.

See Also

-

3.5.6 TSIX_VOUT_Enable

ClientVersion 12, and higher *<license:TBD>*

```
TSI_RESULT __stdcall TSIX_VOUT_Enable
(
    TSI_HANDLE Device,
    TSI_FLAGS Flags
);
```

Synopsis

Enable audio/video output. This function can succeed only if the output status at the time of call is disabled. If the output is disabled, its state can't be modified.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Flags

The flags field is currently ignored, and should be set to NULL (0) when calling.

Result

If the function succeeds, the return value is zero and video output will be enabled for configuration. Please note that future versions of TSI may return non-zero positive values to indicate success.

If the function fails, the return value is a negative value indicating an error code.

See Also

-

3.5.7 TSIX_VOUT_Disable

ClientVersion 12, and higher *<license:TBD>*

```
TSI_RESULT __stdcall TSIX_VOUT_Disable
(
    TSI_HANDLE Device
);
```

Synopsis

Disables the currently selected video/audio output. When the output is disabled, its state can't be modified.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Result

If the function succeeds, the return value is zero. Please note that future versions of TSI may return non-zero positive values to indicate success.

If the function fails, the return value is negative indicating an error code.

See Also

-

3.6 Video Preview functions

3.6.1 TSIX_VPREV_SetWindowHandle

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_VPREV_SetWindowHandle
(
    TSI_HANDLE Device,
    OS_WINDOW_ID Container
);
```

Synopsis

Creates video preview inside the given window. The preview will cover the entire client area of the window. The underlying video technology is selected automatically. The API will automatically show video preview in this window if video input is selected and enabled. To disable video preview, set preview window handle to NULL.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Container

Handle to the window to contain the video preview. One video preview per device: if a video is already being shown in another window, that preview will stop and then start again in the indicated window. If this parameter is NULL, then any existing video preview is stopped.

Result

If the function succeeds, the video preview is enabled and the function returns zero. Please note that future versions may return a non-zero positive value to indicate success.

If the function fails, the return value is a negative error code.

3.7 Audio Preview Functions

3.7.1 TSIX_APREV_SetWindowHandle

ClientVersion 12, and later *No license requirements*

```
TSI_RESULT __stdcall TSIX_APREV_SetWindowHandle
(
    TSI_HANDLE Device,
    HWND Container
);
```

Synopsis

Creates a graphical audio preview component. The graphics implementation in version 1.2 is a very basic spectral analysis display which will likely be improved with later versions. The preview will cover the entire client area of the window. The underlying video technology is selected automatically. The API will automatically show spectral analysis graph of the incoming audio if an audio/video input is selected and enabled. To disable the preview, set preview window handle to NULL.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Container

Handle to the window to contain the video preview. One audio preview per device: If audio is already being shown in another window that preview will stop and then start again in the indicated window. If this parameter is NULL, then any existing audio preview is stopped.

Result

If the function succeeds, the audio preview is enabled and the function returns zero. Please note that future versions may return a non-zero positive value to indicate success.

If the function fails, the return value is a negative error code.

See Also

-

3.7.2 TSIX_APREV_SelectDevice

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_APREV_SelectDevice
(
    TSI_HANDLE Device,
    TSI_AUDIO_DEVICE_ID DeviceID
);
```

Synopsis

Select an audio device for audio preview. The system default audio device will always have the *DeviceID* of zero (0). To disable audio preview, set *DeviceID* negative one (-1).

Parameters

Device

Indicates the device on which the operation is to be carried out.

DeviceID

Identifies the device to use for audio preview.

Result

If the function succeeds, the return value is zero. Please note that future versions may return a non-zero positive value to indicate success.

If the function fails, the return value is a negative error code.

See Also

-

3.8 Test system related functions

3.8.1 TSIX_TS_GetTestCount

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_GetTestCount
(
    TSI_HANDLE Device
);
```

Synopsis

Retrieves the number of tests available on the currently selected device. To get a list of tests, please iterate through the list by calling *TSIX_TS_GetTestInfo* function in a loop.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Result

If the function succeeds, the return value is a positive value (or zero) indicating the number of tests available on the device. If the return value is zero, then there are no tests available on the device at the moment.

If the function fails, the return value is a negative error code.

See Also

3.8.2 TSIX_TS_GetTestInfo

3.8.2 TSIX_TS_GetTestInfo

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_GetTestInfo
(
    TSI_HANDLE Device,
    int TestIndex,
    TSI_TEST_ID *ID,
    char *TestName,
    unsigned int TestNameMaxLength
);
```

Synopsis

Retrieves the test ID value and test name of the test indicted by the *TestIndex* parameter.

Parameters

Device

Indicates the device on which the operation is to be carried out.

TestIndex

Test index value ranging from zero (0) to the value returned by a call to *TSIX_TS_GetTestCount* function minus one.

ID

Pointer to a *TSI_TEST_ID* variable, which will receive the test ID value of the test being identified. This ID value is used to start this test.

TestName

Pointer to a char string which will receive the name of the test being identified. If a string is returned, it is guaranteed to be NULL terminated.

This parameter can be NULL. If this parameter is NULL, then TestNameMaxLength parameter must be zero and no test name is returned.

TestNameMaxLength

Number of bytes available in the TestName array. The recommended minimum size is 128 bytes.

Result

If the function succeeds, the return value is the number of chars required by the full name of the test regardless of the *TestNameMaxLength* parameter. If the returned value is EQUAL or HIGHER than *TestNameMaxLength*, it means that the name was truncated.

If the function fails, the return value is a negative error code.

See Also

3.8.1 TSIX_TS_GetTestCount, 3.8.3 TSIX_TS_GetTestParameterCount

3.8.3 TSIX_TS_GetTestParameterCount

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_GetTestParameterCount
(
    TSI_HANDLE Device,
    TSI_TEST_ID ID
);
```

Synopsis

Retrieves the number of parameters required for a particular test. To read the list of parameters, please iterate through the list by calling the *TSIX_TS_GetTestParameterID* function in a loop.

Important: *This function is not needed for applications that only use fully-known device types*

Parameters

Device

Indicates the device on which the operation is to be carried out.

ID

Identifies the test for which to get the parameter count. This test ID value is retrieved either by using the *TSIX_TS_GetTestInfo* function, or by using a constant value defined in a device specific documentation.

Result

If the function succeeds, the return value is a positive value (or zero) identifying the number of parameters required by the indicated test.

If the function fails, the return value is a negative error code.

See Also

3.8.4 TSIX_TS_GetTestParameterID

3.8.4 TSIX_TS_GetTestParameterID

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_GetTestParameterID
(
     TSI_HANDLE Device,
     TSI_TEST_ID ID,
     int ParamIndex,
     TSI_CONFIG_ID *ParamID
     unsigned int *ParamFlags
);
```

Synopsis

Retrieves base information about a parameter that has a bearing on a test. This function can be used to create a generic GUI, which adapts and extends to tests available on a particular device.

Parameters

Device

Indicates the device on which the operation is to be carried out.

ID

Identifies the test to query.

ParamIndex

Index value ranging from zero (0) to the value returned by a call to *TSIX_TS_GetTestParameterCount* function minus one.

ParamID

Pointer to a TSI_CONFIG_ID variable, which will receive the parameter ID value.

ParamFlags

Pointer to an unsigned int value, which will receive flags that provides additional information about the parameter. See table below for flag bits:

Bit	Define	Description
0	TSI_CI_RO	The parameter is read only.
1	TSI_CI_WO	The parameter is write only.
	TSI_CI_RW	Both bits 0 and 1 are set; Parameter is read+write.
2	TSI_CI_CRO	Parameter is cleared on read. This type of parameter is typically also marked read only.
3	TSI_CI_STROBE	Parameter's value is ignored, the act of writing to it triggers an action. This type of parameter is typically also marked write only.
5	TSI_CI_NOLOAD	The parameter's value is not loaded from a configuration file, even if the parameter value is present in the file.
6	TSI_CI_NOSAVE	The parameter's value is not written into a configuration file.

(Continued...)

(...Continued)

Result

If the function succeeds, the return value is a positive value (or zero) indicating the size of the configuration item in bytes. If the return value is zero, it means that the configuration item's size is not constant, or depends on values contained in other configuration items.

If the function fails, the return value is a negative error code.

See Also

3.8.3 TSIX_TS_GetTestParameterCount

3.8.5 TSIX_TS_Clear

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_Clear
(
    TSI_HANDLE Device
);
```

Synopsis

Resets the test system to its default settings. Please refer to section *6 Tests* for details on the test specific defaults. A device must be selected before calling this function. If no device is selected before calling this function, this function will select the default device.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Result

If the function succeeds, the return value is zero. Please note that future versions may return a non-zero positive value to indicate success.

If the function fails, the return value is a negative error code.

See Also

6 Tests

3.8.6 TSIX_TS_SetConfigItem

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_SetConfigItem
(
     TSI_HANDLE Device,
     TSI_CONFIG_ID ConfigItemID,
     void *ItemData,
     unsigned int ItemSize
);
```

Synopsis

Set a test-system configuration item. If the given configuration ID is valid, the function will copy the client provided data into API internal data storage for later use by the test system. Please refer to 6 Tests and 5 Device Control & Information for details on the configuration items. A device must be selected before calling this function. If no device is selected before calling this function, this function will select the default device.

Parameters

Device
Indicates the device on which the operation is to be carried out.

ConfigItemID
Identifies which configuration item to set.

ItemData
Pointer to the new data-set for the configuration item.

ItemSize
Size of the new data to be set for the configuration item.

Result

If the function succeeds, the return value is zero. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, the return value is a negative error code.

See Also

3.8.7 TSIX_TS_GetConfigItem, 3.8.9 TSIX_TS_LoadConfig

3.8.7 TSIX_TS_GetConfigItem

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_GetConfigItem
(
    TSI_HANDLE Device,
    TSI_CONFIG_ID ConfigItemID,
    void *ConfigItemData,
    unsigned int ItemMaxSize
);
```

Synopsis

Retrieve the current setting of a configuration item. If the ConfigItemID is valid and the provided data buffer is large enough, the function will copy the data to the provided buffer. A device must be selected before calling this function. If no device is selected before calling this function, this function will select the default device.

Parameters

Device
Indicates the device on which the operation is to be carried out.

ConfigItemID
Identifies the configuration item to read.

ConfigItemData
Pointer to a buffer which will receive the configuration item data.

ItemMaxSize
Size of the ConfigItemData buffer in bytes.

Result

If the function succeeds, the return value is the number of bytes required to hold the configuration item data regardless of the ItemMaxSize parameters.

Important: *If the return value is HIGHER than ItemMaxSize parameter it means that no data was actually copied to the ConfigItemData buffer. In this case the contents of the ConfigItemData buffer are unchanged.*

If the function fails, the return value is a negative error code.

See Also

3.8.6 TSIX_TS_SetConfigItem, 3.8.8 TSIX_TS_SaveConfig

3.8.8 TSIX_TS_SaveConfig

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_SaveConfig
(
    TSI_HANDLE Device,
    char *Filename
);
```

Synopsis

Saves the current test system configuration into a file for later use.

Parameters

Device

Indicates the device on which the operation is to be carried out.

FileName

Pointer to a NULL terminated string containing the fully qualified filename of the target file. The API will overwrite any existing file.

Result

If the function succeeds, the return value is a positive, non-zero value indicating the number of bytes written to the target file.

If the function fails, the return value is a negative error code.

See Also

3.8.9 TSIX_TS_LoadConfig

3.8.9 TSIX_TS_LoadConfig

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_LoadConfig
(
    TSI_HANDLE Device,
    char *FileName
);
```

Synopsis

The API will first open the file. If the file was succesfully opened, the API continues to clear the test system configuration and loads new configuration from the given file.

Important: *If the file is corrupted and/or there is problem reading the file the test system state after the function call will be the API default configuration.*

Parameters

Device

Indicates the device on which the operation is to be carried out.

FileName

Pointer to a NULL terminated string containing the fully qualified path of the configuration file.

Result

If the function succeeds, the return value is zero and the test system configuration was loaded from the given file. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, the return value is a negative error code and the test system configuration status in undefined.

See Also

3.8.8 TSIX_TS_SaveConfig

3.8.10 TSIX_TS_RunTest

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_RunTest
(
    TSI_HANDLE Device,
    TSI_TEST_ID TestID
);
```

Synopsis

Run the given test. The function will block the calling application until the test is completed. Please refer to chapter 6 Tests for details on available tests. Depending on type of test being ran, an input or output must be selected and enabled before calling this function.

Parameters

Device

Indicates the device on which the operation is to be carried out.

TestID

Identifies the test to execute.

Result

If the function was completed without resource allocation issues, hardware problems or other OS errors, the return value is a positive value (or zero) indicating the test result. Please see the table below for test result values.

Value	Define	Description
0	TSI_TEST_PASS	The test is completed with “PASS” status.
1	TSI_TEST_FAIL	The test is completed with “FAIL” status.
2	TSI_TEST_NOT_STARTED	The test procedure failed before the test start conditions were met.

Important: *A previous version of this manual stated different values for the test results. This was due to mistake in the document. To avoid mistakes like these, please use the named constants whenever provided.*

If the function failed due to resource allocation issues; hardware problems or other OS errors, the return value is a negative error code.

See Also

6 Tests

3.8.11 TSIX_TS_CaptureReference

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_CaptureReference
(
    TSI_HANDLE Device,
    int RequiredMatches,
    int ReferenceIndex
);
```

Synopsis

Captures a reference frame from the currently selected device and input. The function will block the calling thread until a reference frames is captured, or an error is encountered. An input must be selected and enabled before calling this function.

Important: *The RequiredMatches parameter can be used perform a sanity check in order to select a known good reference frame. If the RequiredMatches parameter is non-zero, the function will require a sequence of identical frames to be captured before accepting a frame as reference. Recommended setting for RequiredMatches is 2 for digital sources. Analog sources should always use 0 (=disable), since analog captures are practically never identical.*

Important: *If RequiredMatches is non-zero, the function will attempt to capture up to 60 frames in order to get a good reference frame. If no acceptable reference frame is captured within the period of 60 frames, the function fails.*

Important: *This function should be used only when the source device is supposed to be sending a static image.*

Parameters

Device
Indicates the device on which the operation is to be carried out.

RequiredMatches
Number of identical frames to be received before accepting the frame as reference. Allowed range is 0 – 10. Zero setting will not do any checking and will accept the first frame captured.

ReferenceIndex
Identifies which reference frame is to be set. This parameter must be zero.

Result

If the function succeeds, the return value is zero and the reference frame and related configuration items are set automatically. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, no reference frame was captured and any previous reference frame configuration remains unchanged.

See Also

3.8.6 TSIX_TS_SetConfigItem, 3.8.9 TSIX_TS_LoadConfig

3.8.12 TSIX_TS_WaitInputSignal

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_TS_WaitInputSignal
(
    TSI_HANDLE Device,
    unsigned int MaxWait
);
```

Synopsis

Blocks the calling thread until video and/or audio signal is detected on the selected and enabled input, or the timeout period has elapsed.

Important: *The function cannot guarantee that the signal is still present after the function has returned.*

Parameters

Device

Indicates the device on which the operation is to be carried out.

MaxWait

Indicates maximum amount of time to wait for input signal to be detected, in milliseconds.

Results

If the function succeeds, and input signal is detected within the given timeout period, the return value is zero.

If the timeout expires before input signal is detected, the return value is TSI_ERROR_TIMEOUT.

If the function fails, the return value will be a negative error code.

3.9 Misc functions

3.9.1 TSIX_MISC_SaveReference

ClientVersion 12, and later *No license requirements*

```
TSI_RESULT __stdcall TSIX_MISC_SaveReference
(
    TSI_HANDLE Device,
    char *FileName,
    unsigned int RefIndex,
    TSI_FRAME_FORMAT_ID FormatID
);
```

Synopsis

Save the current reference frame into a file.

Parameters

Device

Indicates the device on which the operation is to be carried out.

FileName

Pointer to a NULL terminated char string identifying the target file. If the file already exists, it will be overwritten without prompt.

RefIndex

Reference Frame index. This parameter must be zero.

FormatID

Identifies the image file's format. The reference frame data will be converted to be suitable for the given format.

ID	Define	Description
1	TSI_FRAME_FORMAT_BMP	Identifies 24 effective bits per pixel BMP file type. >24 effective bits per pixel images are converted, and data loss is allowed.
2	TSI_FRAME_FORMAT_PPM	Identiefies 24 or 48 effective bits per pixel PPM file type. Lossless format.
3	TSI_FRAME_FORMAT_BMP_LOSSLESS	Identifies 24 effective bits per pixel BMP file. Data l oss is not allowed in conversion.

Result

If the function succeeds, the return value is the size of the resulting image file in bytes.

If the function fails, the return value is a negative error code.

See Also

3.9.2 TSIX_MISC_LoadReference, 3.8.11 TSIX_TS_CaptureReference, 3.8.8 TSIX_TS_SaveConfig

3.9.2 TSIX_MISC_LoadReference

ClientVersion 12, and later *No license requirements*

```
TSI_RESULT __stdcall TSIX_MISC_LoadReference
(
    TSI_HANDLE Device,
    char *FileName,
    unsigned int RefIndex
);
```

Synopsis

Load a reference image from a file.

Parameters

Device

Indicates the device on which the operation is to be carried out.

FileName

Identifies the source file from which to read data.

RefIndex

Reference frame index. This parameter must be zero.

Result

If the function succeeds, the return value is zero and a reference frame is loaded. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, the return value is a negative error code.

See Also

3.9.1 TSIX_MISC_SaveReference, 3.8.9 TSIX_TS_LoadConfig

3.9.3 TSIX_MISC_SetOption

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_MISC_SetOption
(
    TSI_HANDLE Device,
    TSI_OPTION_ID OptionID,
    int OptionValue
);
```

Synopsis

Important: *Currently, there are no options defined, and the function is not used.*

Get and set option value. The function will set the new option value, and return the previous setting to the client application. A device must be selected before calling this function. If no device is selected before calling this function, this function will select the default device.

Important: *If the new option value is negative or out of range for the option in question, the option will remain unchanged and the function returns the current value of the option: Therefore, negative OptionValue parameter transforms the function to only read the current option value without changing it.*

Parameters

Device
: Indicates the device on which the operation is to be carried out.

OptionID
: Identifies the option to get and set.

OptionValue
: Contains the new value for the option. Valid option setting values are positive numbers (or zero).

Result

If the function succeeds, the return value is a positive value (or zero) indicating the previous setting of the option.

If the function fails, the return value is a negative error code.

See Also

-

3.10 Status log functions

3.10.1 TSIX_STLOG_GetMessageCount

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_STLOG_GetMessageCount
(
    TSI_HANDLE Device
);
```

Synopsis

Retrieves the number of queued status log message.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Result

If the function succeeds, the return value is a positive value indicating the number of queued status messages lines. If the return value is zero, then there are no messages queued.

If the function fails, the return value is a negative error code.

See Also

3.10.2 TSIX_STLOG_Clear, 3.10.3 TSIX_STLOG_GetMessageData

3.10.2 TSIX_STLOG_Clear

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_STLOG_Clear
(
    TSI_HANDLE Device
);
```

Synopsis

Clear the status log buffer.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Result

If the function succeeds, the return value is zero. Please note that future versions may return non-zero positive value to indicate success.

If the function fails, the return value is a negative error code.

See Also

-

3.10.3 TSIX_STLOG_GetMessageData

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_STLOG_GetMessageData
(
    TSI_HANDLE Device,
    char *MsgBuffer,
    unsigned int MaxReadSize,
    unsigned int *OutputSize
);
```

Synopsis

Reads status log message buffer data. The function will output only a single, complete line of text without newline character(s) – insert newline character(s) as necessary if writing to a file and/or displaying on screen.

Parameters

Device

Indicates the device on which the operation is to be carried out.

MsgBuffer

Pointer to an array of characters which will receive a single line of status log text. The resulting string is guaranteed to be NULL terminated. The string will not contain newline character(s).

This parameter can be NULL. If this parameter is NULL, the MaxReadSize parameter must be set to zero. If this parameter is NULL, the function will return the MsgBuffer size required to get the next line of status log text.

MaxReadSize

Number of characters allocated for the MsgBuffer. Recommended minimum size for status log message line is 256 characters.

OutputSize

Pointer to an unsigned int variable that will receive the number of characters copied to the MsgBuffer string not counting the terminating NULL character.

If this parameter is NULL, then the number of copied characters is not returned.

Result

If the function succeeds, the return value is zero and the NULL terminated status log string is placed to MsgBuffer.

If the MsgBuffer was not large enough to contain the line of text plus the terminating NULL, the return value is a positive value indicating the required MsgBuffer size. The given MsgBuffer is erased.

If the function fails, the return value is a negative error code and the contents of the MsgBuffer are undefined.

See Also

3.10.1 TSIX_STLOG_GetMessageCount

3.10.4 TSIX_STLOG_WaitMessage

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_STLOG_WaitMessage
(
     TSI_HANDLE Device,
     int MaxWait
);
```

Synopsis

Wait for at least one status log messages to become available for reading. If no messages arrive within given period, the function will return zero.

Parameters

Device

Indicates the device on which the operation is to be carried out.

MaxWait

Maximum time to wait for message(s) to arrive, in milliseconds.

Result

If the function succeeds, the return value is zero, or a positive number indicating the number of readable status log messages lines available for reading.

If the function fails, the return value is a negative error code.

See Also

3.10.1 TSIX_STLOG_GetMessageCount and 3.10.3 TSIX_STLOG_GetMessageData

3.11 Report generator functions

3.11.1 TSIX_REP_BeginLogRecord

ClientVersion 12, and higher *No license requirements*

```
TSI_RESULT __stdcall TSIX_REP_BeginLogRecord
(
    TSI_HANDLE Device,
    char *TargetFile,
    char *DUT_Information
);
```

Synopsis

Starts HTML report generator. The report generator will gather information about the current TE, software versions being used during the testing, and test configurations. The report will also record all test activity and test results. The end result is a HTML formatted report. The intention is that for each tested DUT, a separate report file is generated.

To correctly report a test sequence into a report follow these steps:

- Call this function, and make sure it succeeded.
- Run each of the tests planned for a specific DUT device.
- Call the TSI_REP_EndLogRecord function that will generate the final report file.

Parameters

Device
Indicates the device on which the operation is to be carried out.

TargetFile
A NULL terminated string containing the HTML report file name. The file name preferably includes full path to the file.

DUT_Information
A NULL terminated string containing information about the DUT. The information is embedded into the resulting report file.

Results

If the function succeeds, the return value is zero and any relevant information is being gathered into the report.

If the function fails, the return value is a negative error code and the report file is not being generated.

See Also

3.11.2 TSIX_REP_EndLogRecord

3.11.2 TSIX_REP_EndLogRecord

ClientVersion 12, and higher	*No license requirements*

```
TSI_RESULT __stdcall TSIX_REP_EndLogRecord
(
    TSI_HANDLE Device
);
```

Synopsis

Generate the HTML report file contents and release report generator resources. Call this function to complete a DUT test cycle and get the finalized report file about the tests.

Parameters

Device

Indicates the device on which the operation is to be carried out.

Results

If the function succeeds, the return value is zero and the target HTML report file is finalized.

If the function fails, the return value is a negative error code and the target file's contents are undefined.

See Also

3.11.1 TSIX_REP_BeginLogRecord

4 TYPES AND TEST DEFINITIONS

This chapter describes type definitions and test definitions.

4.1 Types

4.1.1 TSI_VERSION_ID

```
Typedef unsigned int TSI_VERSION_ID;
```

4.1.2 TSI_RESULT

```
Typedef int TSI_RESULT;
```

4.1.3 TSI_DEVICE_ID

```
Typedef unsigned int TSI_DEVICE_ID;
```

4.1.4 TSI_INPUT_ID

```
Typedef unsigned int TSI_INPUT_ID;
```

4.1.5 TSI_FLAGS

```
Typedef int TSI_FLAGS;
```

4.1.6 TSI_AUDIO_DEVICE_ID

```
Typedef unsigned int TSI_AUDIO_DEVICE_ID;
```

4.1.7 TSI_CONFIG_ID

```
Typedef unsigned int TSI_CONFIG_ID;
```

4.1.8 TSI_TEST_ID

```
Typedef unsigned int TSI_TEST_ID;
```

4.1.9 TSI_OPTION_ID

```
Typedef unsigned int TSI_OPTION_ID;
```

4.1.10 TSI_HANDLE

```
Typedef void* TSI_HANDLE;
```

4.2 Error codes

-0: **TSI_SUCCESS**: A generic success indication.

-1: **TSI_ERROR_NOT_INITIALIZED**: The TSI API is not properly initialized for operations.

-2: **TSI_ERROR_COMPATIBILITY_MISMATCH**: The given Client version ID is different from the one provided with first call to TSI_Init(). The client application must always use the same Client version ID. Please make sure that the versions of TSI.C, TSI.H and TSI_Types.h match exactly. The version of these files is listed on the second line of the source code.

-3: **TSI_ERROR_NOT_COMPATIBLE**: The given Client version ID is not supported by the loaded API version. Since API is backward compatible with older applications, it means that the application is built for a later version of the API.

-4: **TSI_ERROR_DLL_NOT_FOUND**: Either the TSI.DLL is not found, or one of the lower level API DLLs was not found. Please try to re-install the TSI software package.

-5: **TSI_ERROR_DLL_VERSION_READ**: A failure has occurred while reading version data from a PE Executable file. The file might be corrupted on unreadable or otherwise not useable.

-6: **TSI_ERROR_OUT_OF_MEMORY**: A generic error message indicating a problem when memory was being allocated. Make sure your application is not leaking memory resources. Also remember that 32-bit process can only allocate up to about 2 GB of RAM – the system reserves part of the max. 4GB address space for itself per process.

-7: **TSI_ERROR_FUNCTION_NOT_FOUND**: A required function was not found in a DLL.

-8: **TSI_ERROR_ACCESS_DENIED**: An operation was attempted that requires a license key to be installed, but the license key is not installed for the device being used.

-9: **TSI_ERROR_NO_REFERENCES**: A reference counted item is already at zero references, or the item is already destroyed and can't have any references.

-10: **TSI_ERROR_DEVICE_INDEX_OUT_OF_RANGE**: No device present with the given device index.

-11: **TSI_ERROR_INVALID_PARAMETER**: One or more of the parameters passed to the function are invalid.

-12: **TSI_ERROR_INPUT_ENABLED**: The requested operation is not available while input is enabled.

-13: **TSI_ERROR_INPUT_DISABLED**: The requested operation is not available while input is disabled.

-14: **TSI_ERROR_INPUT_ENABLE_FAILED**: Failed to enable input due to resource allocation problems.

-15: **TSI_ERROR_OPEN_DEVICE**: Failed to open requested device. Usually happens when a previous application fails to exit properly.

-16: **TSI_ERROR_INPUT_SELECT**: Failed to select input.

-17: **TSI_ERROR_NOT_IMPLEMENTED**: The requested function is not implemented in current version.

-18: **TSI_ERROR_UNEXPECTED_ITEM_SIZE**: Get or Set configuration item function: The configuration item's size was unexpected. (Please refer to configuration item details for correct size information).

-19: **TSI_ERROR_UNSUPPORTED_CONFIG_ID**: Get or Set configuration item function: The given configuration item is not supported with the current hardware, or the ID is unknown.

-20: **TSI_ERROR_CONFIGURATION_ITEM_NOT_SET**: Get configuration item function: The given configuration function has no value assigned to it at the moment.

-21: **TSI_ERROR_FILE_CREATE**: Failed to create save file.

-22: **TSI_ERROR_FILE_WRITE**: Failed to write into a file.

-23: **TSI_ERROR_FILE_OPEN**: Failed to open an existing file.

-24: **TSI_ERROR_FILE_READ**: Failed to read from a file.

-25: **TSI_ERROR_INVALID_FILE**: Unsupported file format.

-26: **TSI_ERROR_DATA_CORRUPTED**: File contents are corrupted or the file is partial.

-27: **TSI_ERROR_FORMAT_MISMATCH**: Reference frame does not match incoming video signal.

-28: **TSI_ERROR_INVALID_TEST_MODE**: Requested testing mode is not supported.

-29: **TSI_ERROR_COMPARE_FAILED**: Test procedure failed – Test outcome is not determined.

-30: **TSI_ERROR_NO_REFERENCE_FRAME**: Can't start test because no reference frames are set.

-31: **TSI_ERROR_TIMEOUT**: An asynchronous operation is taking too long, and was aborted after a timeout event occurred.

-32: **TSI_ERROR_NO_DATA_AVAILABLE**: The requested data is not available. Please note that data can become available without intervention from client software. (For example video timing configuration items).

-33: **TSI_ERROR_CONFIG_ITEM_ACCESS**: Configuration item read or write failed.

-34: **TSI_ERROR_PRESENT_GRAPHICS**: Failed to show graphics preview frame/modify preview area properties.

-35: **TSI_ERROR_UNSUPPORTED_FORMAT**: The captured format is not supported.

-36: **TSI_ERROR_DEVICE_SPECIFIC**: An unexpected problem occurred in the device specific software component.

-37: **TSI_ERROR_DISK_FILE_IO**: A problem occurred while reading or writing to a file.

-38: **TSI_ERROR_INTERNAL**: Internal state is invalid, or some other internal issue.

-39: **TSI_ERROR_CONFIGURATION_ITEM_VALUE**: Attempted to set a configuration item to a value that is not allowed.

-40: **TSI_ERROR_CAPTURE_BROKEN**: Capture (Video, audio or other signal) failed and was re-started during testing.

-41: **TSI_ERROR_OS_ERROR**: An OS function call has failed. Note that some OS function call failures have specific error codes, like TSI_ERROR_FILE_CREATE.

-42: **TSI_ERROR_DATA_PROTECTION_ENABLED**: Video or Audio data is HDCP protected and can't be used for testing or saving.

-43: **TSI_ERROR_TEST_REQUIREMENTS_NOT_MET**: Test requirements were not met by the capture device.

-44: **TSI_ERROR_UNSUPPORTED_COLORSPACE**: The frame color space is not supported by the function

-45: **TSI_ERROR_NO_DEVICE_SELECTED:** No device is currently selected, and the attempted operation requires a device to be selected.

-46: **TSI_ERROR_INVALID_ACCESS_MODE:** Attempted to read a write-only CI, or write a read-only CI.

-47: **TSI_ERROR_OUTPUT_ENABLED:** The attempted operation is not available if source function is enabled.

-48: **TSI_ERROR_OPERATION_DATA_LOSS:** The attempted operation would have caused unwanted data loss. (For example, saving into a file that cannot support the necessary color depth of the reference data).

-49: **TSI_ERROR_BAD_FW_VERSION:** The firmware version on the device being selected is not compatible with the current TSI and/or device software.

5 DEVICE CONTROL & INFORMATION

This section defines how to read information from devices, and how to control device hardware features.

5.1 Generic realtime measurements

```
ClientVersion 11, and higher
```

This section defines Configuration Items used to access generic measurement data for devices that support it.

5.1.1 ADC Data access CI range

```
TSI_R_ADC_FIRST                                 0x12000
TSI_R_ADC_LAST                                  0x120ff
```

Type information

```
unsigned int ADC_Data[0x100]                    ARRAY_U32
0 to 1024 bytes                                 RO
```

Description

The CI range from 0x12000 to 0x120ff maps into a block of up to 256 4-byte data containers. These data available through this data block depends on the used device.

Each CI ID in this range supports variable size read, so it is possible to get complete results structure in one read, starting at a user selectable DWORD offset into the structure.

See TSI_W_USBC_ADC_TIMEOUT below to control whether the function blocks until updated data is available or not.

5.1.2 ADC Data available on UCD-340

See table below for convenience CI ID definitions that make it easy to access certain data from the ADC Data range.

Define	Config ID	Description
TSI_R_ADC_VBUS_VOLTAGE	0x12000	Voltage on Vbus, in millivolts (mV).
TSI_R_ADC_VBUS_CURRENT	0x12001	Current on Vbus, in milliamperes (mA).
TSI_R_ADC_VCC1_VOLTAGE	0x12002	Voltage on CC1, in millivolts (mV).
TSI_R_ADC_VCC2_VOLTAGE	0x12003	Voltage on CC2, in millivolts (mV).
TSI_R_ADC_VCONN_VOLTAGE	0x12004	Voltage on Vconn, in millivolts (mV).
TSI_R_ADC_VCONN_CURRENT	0x12005	Current on Vconn, in milliamperes (mA).

5.1.3 TSI_W_USBC_ADC_TIMEOUT

```
TSI_W_USBC_ADC_TIMEOUT                          0x12101
unsigned int usbc_adc_timeout                   U32
4 bytes                                         WO
```

Synopsis

Controls whether the ADC scanner (see TSI_R_ADC_FIRST above) waits for data to become updated or returns immediately with the last cached values. When TSI_W_USBC_ADC_TIMEOUT is 0 (default), the last cached value is returned. When TSI_W_USBC_ADC_TIMEOUT is nonzero it is the amount of time the function blocks (in ms) waiting for data to be updated. If the call times out, TSI_ERROR_TIMEOUT (-31) is returned.

5.1.4 TSI_W_USBC_ADC_CTRL

```
TSI_W_USBC_ADC_CTRL                             0x12100
unsigned int usbc_adc_ctrl                      U32
4 bytes                                         WO
```

Synopsis

Controls the ADC real time measurement system. Available commands are: 1 = Enable ADC data scanner. 2 = Disable ADC data scanner. The ADC scanner is disabled by default.

5.2 Generic low-level test results

ClientVersion 11, and higher

This section defines Configuration Items used to read low-level test results from various tests.

5.2.1 RAW test results access CI range

TSI_R_TDATA_BLOCK_FIRST	0x0F800
TSI_R_TDATA_BLOCK_LAST	0x0FFD0

Type information

unsigned int TData[0x7d0]	ARRAY_U8
1 to 8000 bytes	RO

Description

The CI range starting from CI ID 0x0F800 up to 0x0FFD0 maps to a 8000 byte memory region with 4 byte access granularity. Any TSI test can return results through this data block in a test specific structure. Please refer to test descriptions / CI information for details per test.

Each CI ID in this range supports variable size read, so it is possible to get complete results structure in one read, starting at a user selectable DWORD offset into the structure. Unused bytes will have the value 0xAF.

5.2.2 TSI_R_TDATA_BLOCK_SIZE

TSI_R_TDATA_BLOCK_SIZE	0x0FFFF
unsigned int TdataRawSize	U32
4 bytes	RO

Description

This CI contains number of bytes of RAW test results data available for reading from CI TSI_R_TDATA_GENERIC_STRUCT_VERSION.

5.2.3 TSI_R_TDATA_GENERIC_STRUCT_VERSION

```
TSI_R_TDATA_GENERIC_STRUCT_VERSION          0x0f800
unsigned int TData                          U32
4 bytes                                     RO
```

Description

Each test results table begins with version information so that applications can detect if the wanted field is available in the results data.

Available after running any test that supports RAW results gathering.

5.2.4 TSI_R_TDATA_USBC_EL_VCC*

```
TSI_R_TDATA_USBC_EL_VCC1                    0x0f801
TSI_R_TDATA_USBC_EL_VCC2                    0x0f803
unsigned int Tdata                          U32
4 bytes                                     RO
```

Description

Contains CC1 and CC2 voltage measurements as millivolts.

Available after running test with ID 0xC0000 (6.6.1 USBC Electrical Test Set / Up Face port CC and Vconn test).

5.2.5 TSI_R_TDATA_USBC_EL_VCONN*

```
TSI_R_TDATA_USBC_EL_VCONN1                  0x0f802
TSI_R_TDATA_USBC_EL_VCONN2                  0x0f804
unsigned int Tdata                          U32
4 bytes                                     RO
```

Description

Contains VConn1 and VConn2 line voltages when in Vconn role as millivolts.

Available after running test with ID 0xC0000 (6.6.1 USBC Electrical Test Set / Up Face port CC and Vconn test).

5.2.6 TSI_R_TDATA_USBC_VAUX1_*

```
TSI_R_TDATA_USBC_VAUX1_P                    0x0f801
TSI_R_TDATA_USBC_VAUX1_N                    0x0f802
unsigned int Tdata                          U32
4 bytes                                     RO
```

Description

Positive (_P) and Negative (_N) voltage levels on AUX1 line when in direct cable mode as millivolts.

Available after running test with ID 0xC0001 (6.6.2 USBC Electrical Test Set / AUX (SBU) lines test).

5.2.7 TSI_R_TDATA_USBC_VAUX2_*

```
TSI_R_TDATA_USBC_VAUX2_P                    0x0f803
TSI_R_TDATA_USBC_VAUX2_N                    0x0f804
unsigned int Tdata                          U32
4 bytes                                     RO
```

Description

Positive (_P) and Negative (_N) voltage levels on AUX2 line when in crossed cable mode as millivolts.

Available after running test with ID 0xC0001 (6.6.2 USBC Electrical Test Set / AUX (SBU) lines test).

5.2.8 TSI_R_TDATA_USBC_EL_VBUS_V

```
TSI_R_TDATA_USBC_EL_VBUS_V                  0x0f801
unsigned int TData                          U32
4 bytes                                     RO
```

Description

Contains Vbus voltage, in millivolts.

Available after running tests with ID 0xC0002 (6.6.3 USBC Electrical Test Set / DUT as Power Sink) or 0xC0003 (6.6.4 USBC Electrical Test Set / DUT as Power Source).

5.2.9 TSI_R_TDATA_USBC_EL_VBUS_I*

```
TSI_R_TDATA_USBC_EL_VBUS_I1                    0x0f802
TSI_R_TDATA_USBC_EL_VBUS_I2                    0x0f803
TSI_R_TDATA_USBC_EL_VBUS_I3                    0x0f804
TSI_R_TDATA_USBC_EL_VBUS_I4                    0x0f805
unsigned int Tdata                             U32
4 bytes                                        RO
```

Description

Contains Vbus current, in milliamps for each of the four Vbus wires separately. To get total current, the values must be added together.

Available after running tests with ID 0xC0002 (6.6.3 USBC Electrical Test Set / DUT as Power Sink) or 0xC0003 (6.6.4 USBC Electrical Test Set / DUT as Power Source).

5.2.10 TSI_R_TDATA_USBC_EL_GND_I*

```
TSI_R_TDATA_USBC_EL_GND_I1                     0x0f806
TSI_R_TDATA_USBC_EL_GND_I1                     0x0f807
TSI_R_TDATA_USBC_EL_GND_I1                     0x0f808
TSI_R_TDATA_USBC_EL_GND_I1                     0x0f809
unsigned int Tdata                             U32
4 bytes                                        RO
```

Description

Contains GND current, in milliamps, for each of the four Vbus ground wires separately. To get total current, the values must be added together.

Available after running tests with ID 0xC0002 (6.6.3 USBC Electrical Test Set / DUT as Power Sink) or 0xC0003 (6.6.4 USBC Electrical Test Set / DUT as Power Source).

5.3 Input video format

The following configuration items define the video format being received by the selected video input and the memory layout selected for it by TSI.

5.3.1 TSI_R_INPUT_WIDTH

```
TSI_R_INPUT_WIDTH                                   0x200
unsigned int InputW                                 U32
4 bytes                                             RO
```

Description

Defines the video frame width as number of elements. To get width as number of pixels, multiply this value with value of TSI_R_INPUT_ELEMENT_WIDTH CI.

5.3.2 TSI_R_INPUT_HEIGHT

```
TSI_R_INPUT_HEIGHT                                  0x201
unsigned int InputH                                 U32
4 bytes                                             RO
```

Description

Defines the video frame height as number of elements. To get height as number of pixels, multiply this value with value of TSI_R_INPUT_ELEMENT_HEIGHT CI.

5.3.3 TSI_R_INPUT_FREQ

```
TSI_R_INPUT_FREQ                                    0x202
unsigned int InputFrequency                         U32
4 bytes                                             RO
```

Description

Defines the frame rate of the input video signal as a fixed point value with scale factor of 10. To get Hz, divide the value with the scale factor.

5.3.4 TSI_R_INPUT_ELEMENT_SIZE

```
TSI_R_INPUT_ELEMENT_SIZE                    0x203
unsigned int InputElemSize                  U32
4 bytes                                     RO
```

Description

Defines the RAM storage size of a single element as number of bytes.

5.3.5 TSI_R_INPUT_ELEMENT_WIDTH

```
TSI_R_INPUT_ELEMENT_WIDTH                   0x204
(TSI_R_INPUT_PIXELS_PER_ELEMENT             0x204)
unsigned int InputElemW                     U32
4 bytes                                     RO
```

Description

Defines the width of a single element as number of pixels.

Important: *The define TSI_R_INPUT_PIXELS_PER_ELEMENT is now considered obsolete, but it remains to be defined for backwards compatibility. The define was changed to better describe the data.*

5.3.6 TSI_R_INPUT_ELEMENT_HEIGHT

```
TSI_R_INPUT_ELEMENT_HEIGHT                  0x205
(TSI_R_INPUT_LINES_PER_ELEMENT              0x205)
unsigned int InputElemH                     U32
4 bytes                                     RO
```

Description

Defines the height of a single element as number of pixels.

Important: *The define TSI_R_INPUT_LINES_PER_ELEMEMENT is now considered obsolete, but it remains to be defined for backwards compatibility. The define was changed to better describe the data.*

5.3.7 TSI_R_INPUT_COLOR_DEPTH

```
TSI_R_INPUT_COLOR_DEPTH                          0x206
unsigned int InputColorDepth                     U32
4 bytes                                          RO
```

Description

Defines the color depth of the input signal. While this value has no effect on the element memory layout, it does indicate how many color bits are expected to be received from the video input.

5.3.8 TSI_R_INPUT_ELEMENT_FORMAT

```
TSI_R_INPUT_ELEMENT_FORMAT                       0x207
(TSI_R_INPUT_PIXEL_FORMAT                        0x207)
unsigned int InputElemFormat                     U32
4 bytes                                          RO
```

Description

Defines the element format used to encode the pixel data of the bitmap. Please see table below for currently defined format ID values:

Define	ID	Description
TSI_ELF_RGB_080808	0	RGB color, max color depth 8 bits per channel. Encoded as 3 unsigned bytes or 4 unsigned bytes per element.
TSI_ELF_RGB_161616	1	RGB color, max color depth 16 bits per channel. Encoded as 3 unsigned shorts or 4 unsigned shorts per element.
TSI_ELF_YCbCr_080808	0x100	YCbCr color, max color depth 8 bits per channel. Encoded as 3 unsigned bytes or 4 unsigned bytes per element.
TSI_ELF_YCbCr_161616	0x101	YCbCr color, max color depth 16 bits per channel. Encoded as 3 unsigned shorts or 4 unsigned shorts per element.

Important: *TSI_R_INPIT_PIXEL_FORMAT name define is considered obsolete, however it continues to be defined for backwards compatibility. The new name was incorporated as it is more descriptive.*

5.3.9 TSI_R_INPUT_INTERLACE

```
TSI_R_INPUT_INTERLACE                            0x208
unsigned int InputInterlaced                     U32
4 bytes                                          RO
```

Description

Indicates if the input signal is interlaced. 0 = Progressive, 1 = Interlaced.

5.4 Input audio format

This section defines configuration items define input audio stream properties.

5.4.1 TSI_R_AUDIO_CHANNELS

```
TSI_R_AUDIO_CHANNELS                    0x220
unsigned int AudioChannels              U32
4 bytes                                 RO
```

Description

Indicates number of active audio channels. Number of channels depends on source device, and typically range from 1 to 8 channels.

5.4.2 TSI_R_AUDIO_SAMPLE_RATE

```
TSI_R_AUDIO_SAMPLE_RATE                 0x221
unsigned int AudioRate                  U32
4 bytes                                 RO
```

Description

Indicates audio sample rate in Hz.

5.4.3 TSI_R_AUDIO_SAMPLE_SIZE

```
TSI_R_AUDIO_SAMPLE_SIZE                 0x222
unsigned int AudioSampleSize            U32
4 bytes                                 RO
```

Description

Indicates audio sample size in bytes.

5.4.4 TSI_CAPTURE_AUDIO_MASK

```
TSI_CAPTURE_AUDIO_MASK                          0x238
unsigned int AudioCapMask                       U32
4 bytes                                         RW
```

Description

This configuration item is supported on devices that cannot detect active audio channels and must rely on client application defining the active channels instead. The value is a bit-mask that defines which audio channels from 0 to 7 are being used. Normally, a 5.1 audio would result to value of 63, while the default value “3” translates to stereo (Channels 0 and 1).

5.5 V-by-One inputs

This section contains definitions for configuration items related to V-by-One inputs.

5.5.1 TSI_VX1_SIGNAL_COLOR_DEPTH

```
TSI_VX1_SIGNAL_COLOR_DEPTH                      0x240
unsigned int VX1_ColorDepth                     U32
4 bytes                                         RW
```

Description

Defines color depth expected on the V-by-One input. 0 = 6 bits per color channel, 1 = 8 bits per color channel, 2 = 10 bits per color channel, 3 = 12 bits per color channel. Default setting is “1” (8 bits per color channel).

5.5.2 TSI_VX1_SIGNAL_CHANNELS_PER_UNIT

```
TSI_VX1_SIGNAL_CHANNELS_PER_UNIT                0x241
unsigned int VX1_ChannelsPerU                   U32
4 bytes                                         RW
```

Description

Defines the number of V-by-One channels to capture per unit. Valid settings are 1, 2, 4 or 8. Actual number of channels is this setting multiplied by the value from TSI_R_UNITS_PRESENT CI. <REF_TODO>. Default value is 8.

Important: *Setting this value also resets <REF_TODO> TSI_VX1_SECTION_COUNT CI to it’s default value of one (1).*

5.5.3 TSI_VX1_SIGNAL_SYNC_MODE

```
TSI_VX1_SIGNAL_SYNC_MODE                    0x242
unsigned int VX1_SyncMode                   U32
4 bytes                                     RW
```

Description

Defines the signal sync mode used. 0 = Data enable, 1 = H-Sync + V-Sync. Default setting is “0” (Data enable).

5.5.4 TSI_VX1_HTPDN_CONTROL

```
TSI_VX1_HTPDN_CONTROL                       0x243
unsigned int VX1_HTPDN                      U32
4 bytes                                     RW
```

Description

Defines behavior of the HTPDN signal. 0 = Normal operation, 1 = Force Low, 2 = Force High. Default setting is “0” (Normal operation).

5.5.5 TSI_VX1_LOCKN_CONTROL

```
TSI_VX1_LOCKN_CONTROL                       0x244
unsigned int VX1_LOCKN                      U32
4 bytes                                     RW
```

Description

Defines the behavior of the LOCKN signal. 0 = Normal operation, 1 = Force Low, 2 = Force High, 3 = Force low after TSI_VX1_LOCKN_DELAY. Default setting is “0” (Normal operation).

5.5.6 TSI_VX1_LOCKN_DELAY

```
TSI_VX1_LOCKN_DELAY                         0x245
unsigned int VX1_LOCKN_Delay                U32
4 bytes                                     RW
```

Description

Delay time in µs. This delay is used only if TSI_VX1_LOCKN_CONTROL is set to “3”. Default setting is 41900 µs.

5.5.7 TSI_VX1_VIDEO_VALID_DELAY

```
TSI_VX1_VIDEO_VALID_DELAY                    0x246
unsigned int VX1_VideoValidDelay             U32
4 bytes                                      RW
```

Description

Delay time in µs. Default setting is 41900 µs.

5.5.8 TSI_VX1_FRAME_COMBINE_METHOD

```
TSI_VX1_FRAME_COMBINE_METHOD                 0x247
unsigned int VX1_CombineMode                 U32
4 bytes                                      RW
```

Description

Defines how the frame is built from the individual V-by-One channels. 0 = Default mode (One from each lane is sequence).

5.5.9 TSI_VX1_SECTION_COUNT

```
TSI_VX1_SECTION_COUNT                        0x248
unsigned int VX1_Sections                    U32
4 bytes                                      RW
```

Description

Number of V-by-One Sections the source is sending. Valid setting is any positive value greater than 1 that produces a zero modulus when dividing V-by-One total channels by the new setting. Total V-by-One channels used can be calculated by multiplying values of configuration items TSI_R_UNITS_PRESENT and TSI_VX1_SIGNAL_CHANNELS_PER_UNIT. Default is 1 section.

Important: *Set this configuration item after setting the configuration item TSI_VX1_SIGNAL_CHANNELS_PER_UNIT, as setting it also sets this CI to it's default value of 1.*

5.6 LVDS Inputs

This section contains definitions related to LVDS inputs.

5.6.1 TSI_LVDS_CHANNELS

```
TSI_LVDS_CHANNELS                                0x2a0
unsigned int LVDS_Channels                       U32
4 bytes                                          RW
```

Description

Defines number of LVDS channels to use per LVDS input. Valid settings are 1, 2 or 4. Default setting is 4 (Quad) LVDS.

5.6.2 TSI_LVDS_SIGNAL_COLOR_DEPTH

```
TSI_LVDS_SIGNAL_COLOR_DEPTH                      0x2a1
unsigned int LVDS_ColorDepth                     U32
4 bytes                                          RW
```

Description

Defines input signal's color depth. 0 = 6 bits per color channel, 1 = 8 bits per color channel, 2 = 10 bits per color channel, 3 = 12 bits per color channel. Default setting is "1" (8 bits per color channel).

5.6.3 TSI_LVDS_MAPPING_MODE

```
TSI_LVDS_MAPPING_MODE                            0x2a2
unsigned int LVDS_MapMode                        U32
4 bytes                                          RW
```

Description

Defines pin mapping mode for an LVDS input. 0 = VESA, 1 = JEIDA. Default setting is "1" (JEIDA).

5.7 Accessing Info frames

This section defines configuration items for accessing Info Frames.

5.7.1 TSI_R_HDMI_INFOFRAME_RANGE_*

```
TSI_R_HDMI_INFOFRAME_RANGE_START              0x11000
TSI_R_HMDI_INFOFRAME_RANGE_END                0x110ff
unsigned char data[]                          ARRAY_U8
Variable size                                 RO
```

Synopsis

The HDMI standard allows for maximum of 256 different info-frames. The CI Space starting at 0x11000 has one CI for each possible info-frame. All CI's within the range 0x11000 to 0x110ff are read-only. To read a specific info-frame, add it's ID value to 0x11000 and read that CI. Each CI in this area has dynamic size, and no validity checks are performed on the received info frames. When attempting to read a specific info-frame, please note that if that info frame is not received (ever), the read may fail with error code -32 (No data available).

Some known info-frames have specific CI definitions available for convenience. Please refer to 5.7.3 Additional Info-frame CI definitions, and update bits for info frame CI names and definitions.

5.7.2 TSI_R_HDMI_INFOFRAME_UPDATE_FLAGS

```
TSI_R_HDMI_INFOFRAME_UPDATE_FLAGS             0x11100
unsigned char Flags[32]                       ARRAY_U8
32 bytes                                      CRO
```

Synopsis

Info-frame updated flags. When an info-frame is received, the bit corresponding to it's raw ID is set – For example, for AVI info-frame (ID = 0x82), the bit 0x82 is set.

The data accessible from the CI is a 256-bit flags field stored as a little-endian 256-bit word. The flags are cleared on read. Please refer to 5.7.3 Additional Info-frame CI definitions, and update bits for info frame CI names and definitions.

5.7.3 Additional Info-frame CI definitions, and update bits

Define	Config ID	Update Bit #	Description
TSI_R_HDMI_INFOFRAME_NULL	0x11000	0	***READ ONLY.*** NULL infoframe.
TSI_R_HDMI_INFOFRAME_ACR	0x11001	1	***READ ONLY.*** Audio Clock Regeneration.
TSI_R_HDMI_INFOFRAME_ASP	0x11002	2	***READ ONLY.*** Audio Sample Packet
TSI_R_HDMI_INFOFRAME_GCP	0x11003	3	***READ ONLY.*** General Control Packet
TSI_R_HDMI_INFOFRAME_ACP	0x11004	4	***READ ONLY.*** Audio Content Protection packet
TSI_R_HDMI_INFOFRAME_ISRC1	0x11005	5	***READ ONLY.*** International Standard Recording Code
TSI_R_HDMI_INFOFRAME_ISRC2	0x11006	6	***READ ONLY.*** International Standard Recording Code
TSI_R_HDMI_INFOFRAME_OBA	0x11007	7	***READ ONLY.*** One Bit Audio sample packet
TSI_R_HDMI_INFOFRAME_DTS	0x11008	8	***READ ONLY.*** DTS audio packet
TSI_R_HDMI_INFOFRAME_HBR	0x11009	9	***READ ONLY.*** High BitRate audio stream packet
TSI_R_HDMI_INFOFRAME_GMP	0x1100a	10	***READ ONLY.*** Gamut Metadata Packet
TSI_R_HDMI_INFOFRAME_VSI	0x11081	129	***READ ONLY.*** Vendor Specific Infoframe
TSI_R_HDMI_INFOFRAME_AVI	0x11082	130	***READ ONLY.*** Auxiliary Video Information infoframe
TSI_R_HDMI_INFOFRAME_SPD	0x11083	131	***READ ONLY.*** Source Product Descriptor infoframe
TSI_R_HDMI_INFOFRAME_AIF	0x11084	132	***READ ONLY.*** Audio Infoframe
TSI_R_HDMI_INFOFRAME_MPEG	0x11085	133	***READ ONLY.*** MPEG Source infoframe
TSI_R_HDMI_INFOFRAME_3D_ASP	0x1100b	11	***READ ONLY.*** 3D Audio Sample Packet
TSI_R_HDMI_INFOFRAME_3D_OBA	0x1100c	12	***READ ONLY.*** 3D One Bit Audio sample packet
TSI_R_HDMI_INFOFRAME_AMP	0x1100d	13	***READ ONLY.*** Audio Metadata Packet
TSI_R_HDMI_INFOFRAME_MST_ASP	0x1100e	14	***READ ONLY.*** Multi-stream Audio Sample Packet
TSI_R_HDMI_INFOFRAME_MST_OBA	0x1100f	15	***READ ONLY.*** Multi-stream One Bit Audio sample packet
TSI_R_HDMI_INFOFRAME_DRM	0x11087	135	***READ ONLY.*** Dynamic Range and Mastering infoframe

5.8 Memory management

```
Available in future release
```

This section defines Configuration Items used for device memory management.

5.8.1 TSI_R_MEMORY_SIZE

```
TSI_R_MEMORY_SIZE                                   0x800
unsigned int memory_size                            U64
8 bytes                                             RO
```

Description

Get maximal available device RAM size in bytes.

5.8.2 TSI_R_MEMORY_MAX_BLOCK_AMOUNT

```
TSI_R_MEMORY_MAX_BLOCK_AMOUNT                       0x801
unsigned int max_block_amount                       U64
8 bytes                                             RO
```

Description

Get maximal amount of memory blocks that could be used for memory allocations.

5.8.3 TSI_W_MEMORY_RESET

```
TSI_W_MEMORY_RESET                                  0x802
unsigned int unused                                 U32
4 bytes                                             WO
```

Description

Resets defined memory layout to default. Please refer to 5.8.4 TSI_MEMORY_LAYOUT for details.

5.8.4 TSI_MEMORY_LAYOUT

```
TSI_MEMORY_LAYOUT                                   0x803
unsigned char memory_layout                         ARRAY_U64
Variable size                                       RW
```

Description

Get or set memory blocks allocation layout. Each memory block is stated with U64 size.

Important: *Writing into TSI_MEMORY_LAYOUT will clear already defined memory layout.*

5.8.5 TSI_MEMORY_BLOCK_INDEX

```
TSI_MEMORY_BLOCK_INDEX                          0x804
unsigned int block_index                        U64
8 bytes                                         RW
```

Description

Get index of currently selected memory block.

Set index of memory block, which is intended to be used.

5.8.6 TSI_W_MEMORY_WRITE

```
TSI_W_MEMORY_WRITE                              0x805
unsigned char memory_data                       ARRAY_U8
Variable size                                   WO
```

Description

Uploads RAW data into device memory. Please refer to 5.8.4 TSI_MEMORY_LAYOUT and 5.8.5 TSI_MEMORY_BLOCK_INDEX for additional details.

5.9 Miscellaneous

This section defines miscellaneous configuration items that do not fit any of the other "categories".

Define	Config ID	Default	Description	Reference
TSI_R_GENERIC_STATUS	0x210	N/A	Report generic device status as status bits.	5.9.1
TSI_R_UNITS_PRESENT	0x211	N/A	Number of units chained.	5.9.2
TSI_W_FORCE_HOT_PLUG_STATE	0x212	N/A	Controls HPD state: 0 = low, 1 = High.	5.9.3
TSI_EDID_TE_INPUT (TSI_CURRENT_SINK_EDID)	0x1100	N/A	Access TE side EDID.	5.9.5
TSI_EDID_TE_OUTPUT	0x1101	N/A	Access DUT side EDID over signal cable.	5.9.6
TSI_VERSION_TEXT	0x80000001	N/A	TSI Version information as text.	5.9.7
TSI_LOG_FILE	0x80000002	N/A	Log file-name for recording logs easily.	5.9.8
TSI_INVALID_CONFIG_ITEM	0xffffffff	N/A	Reserved for invalid CI indication. Do not use.	N/A
TSI_HPD_LENGTH	0x1201	1000	HPD Pulse length for TSI generated pulses.	5.9.9
TSI_GLOBAL_LEGACY_DEVICE_HANDLE	0x80000004	N/A	Handle to the device used with the original TSI functions.	5.9.10
TSI_DEVICE_FIRMWARE_INFO	0x80000005	N/A	Returns firmware information as human readable string	5.9.11
TSI_DEVICE_RELOAD_FW	0x80000006	N/A	Re-load firmware. Supported on UCD-3xx and UCD-4xx	5.9.12
TSI_DEVICE_HW_RESET	0x80000007	N/A	Soft-reset hardware components. Supported on UCD-3xx and UCD-4xx.	5.9.13

5.9.1 TSI_R_GENERIC_STATUS

```
TSI_R_GENERIC_STATUS                          0x210
unsigned int generic_status_bits              U32
4 bytes                                       RO
```

Synopsis

Reserved for generic device status bits. Currently this CI is not implemented.

5.9.2 TSI_R_UNITS_PRESENT

```
TSI_R_UNITS_PRESENT                          0x211
unsigned int chain_count                     U32
4 bytes                                      RO
```

Synopsis

Indicates number of chained capture devices including the master device. Please notice that capture device chaining only works with signal types that are scalable by adding more lanes of same type, such as LVDS and V-by-One.

Important: *This CI is implemented only for devices that support chaining.*

5.9.3 TSI_W_FORCE_HOT_PLUG_STATE

```
TSI_W_FORCE_HOT_PLUG_STATE                   0x212
unsigned int force_hpd                       U32
4 bytes                                      WO
```

Synopsis

Forces the HPD to either asserted or deasserted state. 0 = force deasserted state, 1 = force asserted state.

Important: For interfaces that do not support HPD this CI is either not implemented, or has no effect.

5.9.4 TSI_EDID_STREAM_SELECT

```
TSI_EDID_STREAM_SELECT                       0x1102
unsigned int select_stream                   U32
4 bytes                                      RW
```

Synopsis

Select edid stream to be used with TSI_EDID_TE_INPUT and TSI_EDID_TE_OUTPUT.

5.9.5 TSI_EDID_TE_INPUT

```
TSI_EDID_TE_INPUT                            0x1100
(TSI_CURRENT_SINK_EDID                       0x1100)
unsigned char edid_data[]                    ARRAY_U8
Variable size                                RW
```

Synopsis

Allows access to TE side EDID block(s). For reads, use at least 512 bytes buffer. Write sizes are checked by the TE firmware. Typical requirement for size is multiple of 128 bytes.

5.9.6 TSI_EDID_TE_OUTPUT

```
TSI_EDID_TE_OUTPUT                                  0x1101
unsigned char edid_data[]                           ARRAY_U8
Variable size                                       RW
```

Synopsis

Allows access to DUT side EDID block(s) over connecting signal cable. Read capability is always available, while write capability depends on connected DUT device. For reading, use at least 512 bytes buffer. Write sizes depend on DUT and are not checked by TSI. Typical requirement for size is multiple of 128 bytes.

5.9.7 TSI_VERSION_TEXT

```
TSI_VERSION_TEXT                                    0x80000001
char version_string[]                               ARRAY_S8
Variable size                                       RO
```

Synopsis

Reading returns a NULL terminated ASCII text string containing TSI version information, loaded lower-level API's and their versions, and devices available to TSI through them.

Important: *Firmware versions are typically not available as reading this information requires opening the device.*

5.9.8 TSI_LOG_FILE

```
TSI_LOG_FILE                                        0x800000002
char log_filename[]                                 ARRAY_S8
Variable size                                       WO
```

Synopsis

Log filename for easy recording of status log messages. Default setting is "Empty" (No log recorded). To start recording, set a valid filename as NULL terminated ASCII text string into this CI. TSI will automatically record all status log messages into this file. To stop recording, use NULL as the source string pointer.

```
//Start recording:
char *MyLogFile = "C:\\Temp\\MyLog.txt";
TSI_RESULT Result;
Result = TSI_TS_SetConfigItem(TSI_LOG_FILE, MyLogFile, strlen(MyLogFile));
if(Result < TSI_SUCCESS) { /* Handle error */

/* Your actions that need to be logged */

// Stop recording log:
Result = TSI_TS_SetConfigItem(TSI_LOG_FILE, NULL, 0);
if(Result < TSI_SUCCESS) { /* Handle error */
```

5.9.9 TSI_HPD_LENGTH

```
TSI_HPD_LENGTH                                  0x1201
unsigned int hpd_length                         U32
4 bytes                                         WO
```

Synopsis

Length of TSI generated HPD pulses, in milliseconds. Default setting is 1000ms. Set this to zero to disable HPD pulse generation in TSI.

Important: *Even if the HPD pulse generation is disabled, the TE device's firmware may still generate HPD pulses on certain situations.*

5.9.10 TSI_GLOBAL_LEGACY_DEVICE_HANDLE

```
TSI_GLOBAL_LEGACY_DEVICE_HANDLE                 0x80000004
void *TSI_Handle                                PTR
Variable size.                                  RW
```

Synopsis

Contains the device handle which is used when using the Original TSI functions. Size of this value depends on the TSI Edition being used. For x64 version, the size is 8 bytes, and for x86 version, the size is 4 bytes.

Important: *This CI can be used to switch the device being used through the Original TSI functions. However, this operation should be considered potentially dangerous and great care needs to be taken when using this CI for that purpose.*

5.9.11 TSI_DEVICE_FIRMWARE_INFO

```
TSI_DEVICE_FIRMWARE_INFO                    0x80000005
char *FW_Information                        ARRAY_U8
Variable size                               RO
```

Synopsis

Contains a human readable string containing information about the device firmware configuration.

Below is an example printout from a UCD-323 unit:

```
7037AD960000008C
UX 2.0.1 UF 1.3.1
BF 1.1.3
BN 1.4.3
MC 0.20.93
*Image 0 (Code 4)  MF 0.2.7  MN 1.2.7
*Image 1 (Code 11)  MF 0.1.7  MN 1.2.7
*Image 2 (Code 5)  MF 0.2.6  MN 1.2.7
*Image 3 (Code 8)  MF 0.3.13  MN 1.2.7
FE4 0001 00000102
```

5.9.12 TSI_DEVICE_RELOAD_FW

```
TSI_DEVICE_RELOAD_FW                        0x80000006
unsigned int not_used                       U32
4 bytes                                     STROBE
```

Synopsis

If supported, write into this CI triggers firmware reload. This operation is supported on UCD-3xx and UCD-4xx. The value being written is ignored and not used.

5.9.13 TSI_DEVICE_HW_RESET

```
TSI_DEVICE_HW_RESET                          0x80000007
unsigned int reset_code                      U32
4 bytes                                      WO
```

Synopsis

Writing this CI will reset hardware modules as defined by the given reset code. The reset codes may be defined differently for different devices.

Regardless on the type of reset performed on the device, it is up to the user to re-configure the device as needed after any reset. Also, if the USB Front-end controller is reset, the PC ↔ Device communications may break down, in which case it will be necessary to close the device, re-scan for devices, re-open the device and re-configure it before continuing with normal operation.

Please see defined values in table below:

Reset code	Hardware	Description
0	UCD-3xx, UCD-4xx	No effect.
1	UCD-3xx, UCD-4xx	Reset USB Front end controller. ***Important:*** *This component is responsible for handling the PC ↔ Device USB communication, so resetting it may cause the USB communications to break down.* ***Important:*** *This will also reset the USB Front end ↔ Main board communications controller (same as reset code 2).*
2	UCD-3xx, UCD-4xx	Reset USB Front end ↔ Main board communications controller. ***Important:*** *This component handles the communications over the main data bus between the USB board and the Main board. Resetting it should not cause major problems*
3	UCD-3xx	Reset Main board controller.
*	*	RESERVED.

Important: *The intention of this CI is to allow resurrecting the device without power cycle in case the on-board firmware crashes. However, this is still a "soft-reset", and if the USB endpoint controller has failed and/or crashed, it may be impossible to invoke any of these resets.*

Important: *Regardless on the type of reset performed on the device, it is up to the user to re-configure the device as needed after a reset. The recommended way is to treat resets similarly to power-cycling the device: For best results, close and re-open the device after a reset.*

5.10 HDMI Sink, Link Status and Control

HDMI Sink (RX) has the following configuration items to check the status of the HDMI link, and to control it's features:

Name	Description	Reference
TSI_R_HDRX_LINK_STATUS	HDMI RX Link status	5.10.1
TSI_W_HDRX_LINK_CONTROL	HDMI RX Link control	5.10.2
TSI_R_HDRX_ARC_STATUS	HDMI RX Audio Return Channel status	5.10.3
TSI_W_ARC_CONTROL	HDMI RX Audio Return Channel controls	5.10.4

5.10.1 TSI_R_HDRX_LINK_STATUS

```
TSI_R_HDRX_LINK_STATUS                          0x810
unsigned int HDRX_link_status                   U32
4 bytes                                         RO
```

Synopsis

HDMI Link status bits. See bit definitions below:

Bits	Description	
0	TMDS clock detection.	
	0	TMDS Clock not detected
	1	TMDS Clock detected.
1	Clock rate	
	0	Clock rate is 3G.
	1	Clock rate is 6G
2	Input stream lock status	
	0	Not locked.
	1	Locked
3	Link mode	
	0	HDMI mode
	1	DVI mode
6:4	Line lock bits.	
7	HPD status	
31:8	RESERVED	

5.10.2 TSI_W_HDRX_LINK_CONTROL

```
TSI_W_HDRX_LINK_CONTROL                    0x811
unsigned int HDRX_link_control             U32
4 bytes                                    WO
```

Synopsis

Control the HDMI HPD singal. Writing zero (0) will de-assert the HDP signal, and writing one (1) will assert the HPD signal.

5.10.3 TSI_R_HDRX_ARC_STATUS

```
TSI_R_HDRX_ARC_STATUS                      0x812
unsigned int HDRX_arc_status               U32
4 bytes                                    RO
```

Synopsis

Allows access to current ARC status information. See bit definitions below:

Bits	Description
0	ARC Supported. 1 = Supported, 0 = Not supported
1	Internal ARC Audio generator is available. 1 = Available, 0 = Not available.
2	Loop back ARC support. 1 = Supported, 0 = Not supported. If bits 10:8 define no audio sources, it means that the HDMI input itself is used as the loop back source.
7:3	RESERVED
8	Loop back from DVI RX port supported. 1 = Supported, 0 = Not supported.
9	Loop back from DP RX port supported. 1 = Supported, 0 = Not supported.
10	Loop back from SPDIF RX port supported. 1 = Supported, 0 = Not supported.
30:11	RESERVED
31	ARC Enabled. 1 = ARC Is currently enabled, 0 = ARC Is disabled.

5.10.4 TSI_W_ARC_CONTROL

```
TSI_W_ARC_CONTROL                                   0x1210
unsigned int arc_control                            U32
4 bytes                                             WO
```

Synopsis

Used to set-up and control ARC feature on HDMI Inputs. See table below for bit definitions:

Bits	Description	
1:0	ARC State setting	
	0	Disable ARC
	1	Enable and use internal audio generator as audio source.
	2	Enable and use loop back audio source.
7:2	RESERVED	
9:8	Loop back audio source selection	
	0	Loop back audio received from HDMI RX Port
	1	Loop back audio received from DVI RX Port
	2	Loop back audio received from DP RX Port
	3	Loop back audio received from SPDFI RX Port
15:10	RESERVED	
16	ARC Data mode	
	0	Common mode (Utility line + HPD)
	1	Single mode (Utility line only)
31:17	RESERVED	

5.11 HDMI Source, Link Status and Control

HDMI Source (TX) has the following configuration items to check the status of the HDMI link, and to control it's features:

Name	Description	Reference
TSI_W_SRC_HDMI_CONTROL	HDMI Source link controls	5.11.1
TSI_R_SRC_HDMI_STATUS	HDMI Source link status	5.11.2
TSI_R_SRC_HDMI_DUT_CAPS	HDMI Sink device's capabilities	5.11.3

5.11.1 TSI_W_SRC_HDMI_CONTROL

```
TSI_W_SRC_HDMI_CONTROL                          0x800
unsigned int HDMI_src_control                   U32
4 bytes                                         WO
```

Synopsis

Control HDMI Source link features. See table below for control bit definitions:

Bits	Description
0	Scrambler setting. 1 = Enable scrambler, 0 = Disabled scrambler
1	Link speed setting. 1 = 6G Link speed, 0 = 3G Link speed
2	HDMI/DVI mode select. 0 = HDMI Mode, 1 = DVI Mode.
3	HDMI Source behavior. 0 = HDMI 2.0 Source behavior, 1 = HDMI 1.4 Source behavior
31:4	RESERVED

5.11.2 TSI_R_SRC_HDMI_STATUS

```
TSI_R_SRC_HDMI_STATUS                          0x801
unsigned int HDMI_src_status                   U32
4 bytes                                        RO
```

Synopsis

Access HDMI Source link status information. See table below for bit definitions:

Bits	Description
0	Scrambler state. 1 = Scrambler enabled, 0 = Scrambler disabled.
1	Link speed. 1 = 6G, 0 = 3G
2	HDMI/DVI Mode. 1 = DVI Mode, 0 = HDMI Mode.
3	Video enable. 1 = Video is enabled, 0 = Video is disabled
4	Source behavior. 1 = HDMI 1.4 Source behavior, 0 = HDMI 2.0 Source behavior.
5	HPD signal status. 1 = HPD is asserted, 0 = HPD is de-asserted
31:6	RESERVED

5.11.3 TSI_R_SRC_HDMI_DUT_CAPS

```
TSI_R_SRC_HDMI_DUT_CAPS                        0x802
unsigned int HDMI_src_dut_caps                 U32
4 bytes                                        RO
```

Synopsis

Access HDMI DUT capability information. See table below for bit definitions:

Bits	Description
0	SCDC Support. 1 = SCDC is supported by DUT, 0 = SCDC not supported by DUT.
1	Scrambler support. 1 = Scrambler is supported by DUT, 0 = Scrambler is not supported by DUT.
31:2	RESERVED

5.11.4 TSI_SRC_HDMI_SCDC_ADDRESS

```
TSI_SRC_HDMI_SCDC_ADDRESS                      0x803
unsigned int HDMI_scdc_address                 U32
4 bytes                                        RW
```

Synopsis

SCDC address selector. SCDC data will be read or written to this register.

5.11.5 TSI_SRC_HDMI_SCDC_DATA

```
TSI_SRC_HDMI_SCDC_DATA                          0x804
unsigned int HDMI_scdc_data                     U32
32 bytes                                        RW
```

Synopsis

SCDC data from address selected by TSI_SRC_HDMI_SCDC_ADDRESS register. Each read/write of a 32 bit word contains one byte in lowest eight bits.

5.12 HDCP Debugging configuration items

The HDCP debugging is divided into two groups of configuration items, and the 0x280 – 0x2ff configuration item ID range is reserved for the HDCP debug system.

In order to access HDCP Debugging features, a *TSI Advanced license* is required. In addition, accessing HDCP 2.X debugging features, an *HDCP 2.2 license* is required.

First group is for HDCP 1.x, ranging from CI ID 0x280 to 0x28f. Unlisted CI's are reserved for future additions. The new CI's for this group are listed below:

Name	Description	Reference
TSI_R_HDCP_1X_STATUS	HDCP 1.x status indicator bits.	5.12.1
TSI_W_HDCP_1X_COMMAND	HDCP 1.x related commands.	5.12.2

The second CI group is for HDCP 2.x, ranging from 0x290 to 0x29f. Unlisted CI's are reserved for future additions. The new CI's are listed below:

TSI_R_HDCP_2X_STATUS	HDCP 2.x status indicator bits.	5.12.3
TSI_W_HDCP_2X_COMMAND	HDCP 2.x related commands.	5.12.4

In addition, a single other general purpose CI is added (0x212) to allow client applications to generate HPD pulses.

Name	Description	Reference
TSI_W_FORCE_HOT_PLUG_STATE	HPD control of current input	5.12.5

5.12.1 TSI_R_HDCP_1X_STATUS

```
TSI_R_HDCP_1X_STATUS                          0x280
unsigned int hdcp_1x_status;                  U32
4 bytes                                       RO
```

Description

Current status indication of the HDCP 1.x. Please note that this CI will deliver information for both SOURCE and SINK if they are both available on the same device.

The table below defines status bits for HDCP 1.x status on SINK side:

Bits	Field name	Value	Description
0	H1_ACTIVE	0	HDCP 1.x Encryption not active.
		1	HDCP 1.x Encryption is active.
2:1	H1_KEYLD	0	HDCP 1.x Keys not loaded.
		1	HDCP 1.x Test keys loaded.
		2	HDCP 1.x production keys loaded.
		3	RESERVED
3	H1_ENA	0	Current HDCP 1.x indication “not supported” by device.
		1	Current HDCP 1.x indication “supported” by device.
4	H1_AUTH	0	HDCP 1.x Link not authenticated.
		1	HDCP 1.x Link authenticated.
7:5		0	RESERVED
15:4		*	RESERVED for source side HDCP status bits.
16	H1_SUPPORTED	0	HDCP 1.x support is not available on the hardware.
		1	HDCP 1.x support is available on the hardware
17	H1_KEYS	0	HDCP 1.x production use keys are not available
		1	HDCP 1.x production use keys are available with command ID 2
18	H1_TEST_KEYS	0	HDCP 1.x test keys are not available
		1	HDCP 1.x test keys are available with command ID 1
23:19		0	RESERVED
31:24		*	RESERVED for source side HDCP status bits.

(Continued...)

(...Continued)

The table below defines status bits for HDCP 1.x status on SOURCE side:

Bits	Field name	Value	Description
7:0		*	RESERVED for sink side HDCP status bits.
8	H1_ACTIVE	0	HDCP 1.x Encryption not active
		1	HDCP 1.x Encryption is active
10:9	H1_KEYLD	0	HDCP 1.x Keys not loaded.
		1	HDCP 1.x Test keys loaded.
		2	HDCP 1.x production keys loaded.
		3	RESERVED
11	H1_AUTH_PROG	0	Idle
		1	HDCP 1.x authentication in progress
12	H1_AUTH	0	Link not authenticated
		1	Link is HDCP 1.x authenticated.
15:13		0	RESERVED
23:16		*	RESERVED for sink side HDCP status bits.
24	H1_SUPPORTED	0	HDCP 1.x support is not available on the hardware.
		1	HDCP 1.x support is available on the hardware
25	H1_KEYS	0	HDCP 1.x production use keys are not available
		1	HDCP 1.x production use keys are available with command ID 2
26	H1_TEST_KEYS	0	HDCP 1.x test keys are not available
		1	HDCP 1.x test keys are available with command ID 1
31:27		0	RESERVED

5.12.2 TSI_W_HDCP_1X_COMMAND

```
TSI_W_HDCP_1X_COMMAND                          0x281
unsigned int hdcp_1x_command;                  U32
4 bytes                                        WO
```

Description

Write command ID to issue commands to the device to control HDCP 1.x functionality.

Important: *This CI is used to command both source and sink. Available commands are listed below:*

Value	Source / Sink	Description
1	Sink	Load test keys
2	Sink	Load production keys
3	Sink	Unload keys
4	Sink	Set HDCP capable flag
5	Sink	Clear HDCP capable flag
257	Source	Load test keys.
258	Source	Load production keys.
259	Source	Unload production keys.
260	Source	Authenticate.
261	Source	Set state to not authenticated.
262	Source	Enable encryption.
263	Source	Disable encryption.

Important: *Unlisted command ID values are reserved and should not be used.*

5.12.3 TSI_R_HDCP_2X_STATUS

```
TSI_R_HDCP_2X_STATUS                          0x290
unsigned int hdcp_2x_status;                  U32
4 bytes                                       RO
```

Description

Current status indication of the HDCP 2.x. Please note that this CI will deliver information for both SOURCE and SINK if they are both available on the same device.

The following table defines status bits for HDCP 2.x on SINK side:

Bits	Field name	Value	Description
0	H2_ACTIVE	0	HDCP 2.x Encryption not active.
		1	HDCP 2.x Encryption is active.
2:1	H2_KEYLD	0	HDCP 2.x Keys not loaded.
		1	RESERVED
		2	HDCP 2.x production keys loaded.
		3	RESERVED
3	H2_ENA	0	Current HDCP 2.x indication “not supported” by device.
		1	Current HDCP 2.x indication “supported” by device.
4	H2_AUTH	0	HDCP 2.x Link not authenticated.
		1	HDCP 2.x Link authenticated.
7:5		0	RESERVED
15:8		*	RESERVED for HDCP 2.x source side status bits.
16	H2_SUPPORTED	0	HDCP 2.x support is not available on the hardware.
		1	HDCP 2.x support is available on the hardware
17	H2_KEYS	0	HDCP 2.x production use keys are not available
		1	HDCP 2.x production use keys are available with command ID 2
23:18		0	RESERVED
31:24		*	RESERVED. For HDCP 2.x source side status bits.

(Continued...)

(...Continued)

The following table defines status bits for HDCP 2.x on SOURCE side:

Bits	Field name	Value	Description
7:0		*	RESERVED for sink side HDCP status bits.
8	H2_ACTIVE	0	HDCP 2.x Encryption not active
		1	HDCP 2.x Encryption is active
10:9	H2_KEYLD	0	HDCP 2.x Keys not loaded.
		1	RESERVED
		2	HDCP 2.x production keys loaded.
		3	RESERVED
11	H2_AUTH_PROG	0	Idle
		1	HDCP 2.x authentication in progress
12	H2_AUTH	0	Link not authenticated
		1	Link is HDCP 2.x authenticated.
13	H2_KM_STORED	0	KM not stored
		1	KM Stored
15:14		0	RESERVED
23:16		*	RESERVED for sink side HDCP status bits.
24	H2_SUPPORTED	0	HDCP 2.x support is not available on the hardware.
		1	HDCP 2.x support is available on the hardware
25	H2_KEYS	0	HDCP 2.x production use keys are not available
		1	HDCP 2.x production use keys are available with command ID 2
26	H2_TEST_KEYS	0	HDCP 2.x test keys are not available
		1	HDCP 2.x test keys are available with command ID 1
27	KM_SUPPORT_STORE	0	KM Store not supported
		1	KM Store supported
31:28		0	RESERVED

5.12.4 TSI_W_HDCP_2X_COMMAND

```
TSI_W_HDCP_2X_COMMAND                          0x291
unsigned int hdcp_2x_command;                  U32
4 bytes                                        WO
```

Description

Write command ID to issue commands to the device to control HDCP 2.x functionality. Available commands are listed below:

Important: *This CI is used to command both source and sink.*

Value	Source / Sink	Description
2	Sink	Load production keys
3	Sink	Unload keys
4	Sink	Set HDCP capable flag
5	Sink	Clear HDCP capable flag
258	Source	Load production keys.
259	Source	Unload production keys.
260	Source	Authenticate.
261	Source	Set state to not authenticated.
262	Source	Enable encryption.
263	Source	Disable encryption.

Important: *Unlisted command ID values are reserved and should not be used.*

5.12.5 TSI_W_FORCE_HOT_PLUG_STATE

```
TSI_W_FORCE_HOT_PLUG_STATE                     0x212
unsigned int hpd_state;                        U32
4 bytes                                        WO
```

Description

Writing this register will force the HPD status to the indicated state until another state is forced, or TSI generates a HPD pulse for another reason.

To force HPD to low state, write zero. To force HPD to high state, write one.

Important: *Writing this CI does not stop TSI from issuing HPD state changes at a later time. To stop TSI from generating HPD pulses by itself, set the TSI_HDP_LENGTH (0x1201) CI to zero.*

5.13 DP Sink – Link status

The DP Link status access is divided into two groups: Sink current link status and Sink link capabilities. The configuration item ID range 0x2B0 – 0x2EF is reserved for these features. Undefined CI's are reserved for future expansions.

The first group is DP Sink link status data: 0x2B0 - 0x2BF

Name	Description	Reference
TSI_R_DPRX_LINK_STATUS_FLAGS	DP Link status flags	5.13.1
TSI_R_DPRX_LT_STATUS_FLAGS	Previous DP Link training results	5.13.2
TSI_R_DPRX_LINK_VOLTAGE_SWING	DP Link voltage swing data	5.13.3
TSI_R_DPRX_LINK_PRE_EMPHASIS	DP Link pre-emphasis data	5.13.4
TSI_R_DPRX_LINK_LANE_COUNT	DP Link lane count	5.13.5
TSI_R_DPRX_LINK_RATE	DP Link rate	5.13.6
TSI_R_DPRX_ERROR_COUNTS	DP lane error counters	5.13.7
TSI_W_DPRX_DPCD_BASE	DP DPCD base pointer for DPCD register access	5.13.8
TSI_DPRX_DPCD_DATA	For reading and writing DPCD data.	5.13.9
TSI_R_DPRX_CRC_R	Get red CRC value for current stream.	5.13.10
TSI_R_DPRX_CRC_G	Get green CRC value for current stream.	5.13.11
TSI_R_DPRX_CRC_B	Get blue CRC value for current stream.	5.13.12

The second group is DP Sink link capabilities: 0x2C0 - 0x2CF

Name	Description	Reference
TSI_DPRX_MAX_LANES	DP Link max. lanes supported	5.14.1
TSI_DPRX_MAX_LINK_RATE	DP Link max. rate	5.14.2
TSI_DPRX_LINK_FLAGS	DP Link feature flags	5.14.3
TSI_DPRX_STREAM_SELECT	Get or set currently selected stream.	5.14.4

5.13.1 TSI_R_DPRX_LINK_STATUS_FLAGS

```
TSI_R_DPRX_LINK_STATUS_FLAGS                    0x2B0
unsigned int dprx_link_status                   U32
4 bytes                                         RO
```

Description

Indicates current link flags as defined below

Bits	Field name	Value	Description
0	L0_CR		Clock recovery done for lane 0
1	L0_EQ		Channel EQ done for lane 0
2	L0_SL		Symbol lock for lane 0
3		*	RESERVED
4	L1_CR		Clock recovery done for lane 1
5	L1_EQ		Channel EQ done for lane 1
6	L1_SL		Symbol lock for lane 1
7		*	RESERVED
8	L2_CR		Clock recovery done for lane 2
9	L2_EQ		Channel EQ done for lane 2
10	L2_SL		Symbol lock for lane 2
11		*	RESERVED
12	L3_CR		Clock recovery done for lane 3
13	L3_EQ		Channel EQ done for lane 3
14	L3_SL		Symbol lock for lane 3
15		*	RESERVED
16	ILA		Inter-lane align status
17	FRAMING	0	Normal framing
		1	Enhanced framing
18	SCRAMBLING	0	Scrambling disabled
		1	Scrambling enabled
19	FORMAT	0	Single stream mode
		1	Multi stream mode
23:20		*	RESERVED
27:24	STREAMS	*	Number of streams being received at the time of read.
31:28		*	RESERVED

5.13.2 TSI_R_DPRX_LT_STATUS_FLAGS

```
TSI_R_DPRX_LT_STATUS_FLAGS                    0x2B1
unsigned int dprx_lt_status                   U32
4 bytes                                       RO
```

Description

Link status as achieved during previous link-training.

Bits	Field name	Value	Description
0	L0_CR		Clock recovery done for lane 0
1	L0_EQ		Channel EQ done for lane 0
2	L0_SL		Symbol lock for lane 0
3		*	RESERVED
4	L1_CR		Clock recovery done for lane 1
5	L1_EQ		Channel EQ done for lane 1
6	L1_SL		Symbol lock for lane 1
7		*	RESERVED
8	L2_CR		Clock recovery done for lane 2
9	L2_EQ		Channel EQ done for lane 2
10	L2_SL		Symbol lock for lane 2
11		*	RESERVED
12	L2_CR		Clock recovery done for lane 2
13	L2_EQ		Channel EQ done for lane 2
14	L2_SL		Symbol lock for lane 2
31:15		*	RESERVED

5.13.3 TSI_R_DPRX_LINK_VOLTAGE_SWING

```
TSI_R_DPRX_LINK_VOLTAGE_SWING                    0x2B2
unsigned int dprx_link_voltage_swing             U32
4 bytes                                          RO
```

Description

Indicates DP link voltage swing for all active lanes.

Bits	Field name	Value	Description
7:0	L0_VS	0	Lane 0 voltage swing 400 mVpp
		1	Lane 0 voltage swing 600 mVpp
		2	Lane 0 Voltage swing 800 mVpp
		3	Lane 0 Voltage swing 1200 mVpp
		255-4	RESERVED
15:8	L1_VS	*	Voltage swing for Lane 1 (See lane 0 for value definitions)
23:16	L2_VS	*	Voltage swing for Lane 2 (See lane 0 for value definitions)
31:24	L3_VS	*	Voltage swing for Lane 3 (See lane 0 for value definitions)

5.13.4 TSI_R_DPRX_LINK_PRE_EMPHASIS

```
TSI_R_DPRX_LINK_PRE_EMPHASIS                     0x2B3
unsigned int dprx_link_pre_emphasis              U32
4 bytes                                          RO
```

Description

Indicates DP link pre-emphasis for all active lanes.

Bits	Field name	Value	Description
7:0	L0_PE	0	Lane 0 pre-emphasis 0 dB
		1	Lane 0 pre-emphasis 3.5 dB
		2	Lane 0 pre-emphasis 6 dB
		3	Lane 0 pre-emphasis 9.5 dB
		255-4	RESERVED
15:8	L1_PE	*	Pre-emphasis for Lane 1 (See lane 0 for value definitions)
23:16	L2_PE	*	Pre-emphasis for Lane 2 (See lane 0 for value definitions)
31:24	L3_PE	*	Pre-emphasis for Lane 3 (See lane 0 for value definitions)

5.13.5 TSI_R_DPRX_LINK_LANE_COUNT

```
TSI_R_DPRX_LINK_LANE_COUNT                    0x2B4
unsigned int dprx_link_lanes                  U32
4 bytes                                       RO
```

Description

Indicates number of currently active lanes. Valid values are 1, 2 or 4.

5.13.6 TSI_R_DPRX_LINK_RATE

```
TSI_R_DPRX_LINK_RATE                          0x2B5
unsigned int dprx_link_rate                   U32
4 bytes                                       RO
```

Description

Indicates the current link rate as multiple of 0.27Gbps. Typical values are 6, 10 or 20. Please note that some DP standards allow additional link rates.

5.13.7 TSI_R_DPRX_ERROR_COUNTS

```
TSI_R_DPRX_ERROR_COUNTS                       0x2B8
unsigned int dprx_error_data[]                ARRAY_U32
16 bytes                                      CRO
```

Description

Contains error counts for each used DP Lane. Reading the counters also clears them.

Index	Description
0	Error counter for lane 0
1	Error counter for lane 1
2	Error counter for lane 2
3	Error counter for lane 3

5.13.8 TSI_W_DPRX_DPCD_BASE

```
TSI_W_DPRX_DPCD_BASE                          0x2B9
unsigned int dprx_dpcd_base_ptr               U32
4 bytes                                       WO
```

Description

DPCD read and write start pointer into the DPCD register space. The DPCD address value may not exceed 0x00FFFFFF. Default value is 0.

5.13.9 TSI_DPRX_DPCD_DATA

```
TSI_DPRX_DPCD_DATA                            0x2BA
unsigned char dprx_dpcd_data[]                ARRAY_U8
Variable size                                 RW
```

Description

Read and/or write DPCD registers. Each DPCD register is one byte (8 bits). Read/Write access size is not limited, but access size + DPCD base address may not exceed 0x01000000. Please refer to DP Specifications in order to decode the DPCD register data.TSI_R_DPRX_CRC_R

5.13.10 TSI_R_DPRX_CRC_R

```
TSI_R_DPRX_CRC_R                              0x2C4
unsigned int dprx_crc_red                     U32
4 bytes                                       R
```

Description

Get red CRC value for current input video stream. When retrieving CRC values for red, green and blue components, the red component must be read first.

5.13.11 TSI_R_DPRX_CRC_G

```
TSI_R_DPRX_CRC_G                              0x2C5
unsigned int dprx_crc_green                   U32
4 bytes                                       R
```

Description

Get green CRC value for current input video stream. When retrieving CRC values for green and blue components, the red component must be read first.

5.13.12 TSI_R_DPRX_CRC_B

```
TSI_R_DPRX_CRC_B                                0x2C6
unsigned int dprx_crc_blue                      U32
4 bytes                                         R
```

Description

Get blue CRC value for current input video stream. When retrieving CRC values for green and blue components, the red component must be read first.

5.14 DP Sink - Capabilities

5.14.1 TSI_DPRX_MAX_LANES

```
TSI_DPRX_MAX_LANES                          0x2C0
unsigned int dprx_max_lanes                 U32
4 bytes                                     RW
```

Description

Defines maximum number of lanes for link training. Valid settings are 1, 2 and 4.

Important: *Trigger link training by using the TSI_W_FORCE_HOT_PLUG_STATE CI to generate a HPD signal.*

5.14.2 TSI_DPRX_MAX_LINK_RATE

```
TSI_DPRX_MAX_LINK_RATE                      0x2C1
unsigned int dprx_max_link_rate             U32
4 bytes                                     RW
```

Description

Defines maximum link rate for link training. Setting is multipler for 0.27 Gbps. Typical settings are 6 (RBR), 10 (HBR) and 20 (HBR2).

Important: *Trigger link training by using the TSI_W_FORCE_HOT_PLUG_STATE CI to generate a HPD signal.*

5.14.3 TSI_DPRX_LINK_FLAGS

```
TSI_DPRX_LINK_FLAGS                                0x2C2
unsigned int dprx_link_flags                       U32
4 bytes                                            RW
```

Description

Defines which features are indicated as supported for link training. See table below for flag defitions:

Bits	Field name	Value	Description
0	MST	0	Indicate SST support only.
		1	Indicate MST supported.
1	TPS3	0	Indicate TPS3 feature is not supported.
		1	Indicate TPS3 feature is supported
		Important: *For HBR2 (Link rate = 20) capable devices, the TPS3 feature must be indicated as supported (1).*	
31:2		*	RESERVED

Important: *Trigger link training by using the TSI_W_FORCE_HOT_PLUG_STATE CI to generate a HPD signal.*

5.14.4 TSI_DPRX_STREAM_SELECT

```
TSI_DPRX_STREAM_SELECT                             0x2C3
unsigned int dprx_stream_select                    U32
4 bytes                                            RW
```

Description

Get or set currently selected stream. The setting can take effect only while MST is enabled by the source. Any stream can be selected regardless of it’s presence as long as the sink side hardware can receive it.

Important: *The selection is cleared if the source side switches to SST mode.*

5.15 DP Source – Link setup

ClientVersion 11, and higher *<license:TBD>*

This section lists all CI's that are available for reading status or setting parameters for DisplayPort source capable interfaces. Display TX related CI's are allocated in renage from 0x780 to 0x7ff.

5.15.1 TSI_DPTX_LINK_CFG_LANES

```
TSI_SRC_DPTX_CFG_LANES                    0x780
(TSI_SRC_DPTX_LINK_CFG_LANES              0x780)
unsigned int dp_src_lanes                 U32
4 bytes                                   RW
```

Synopsis

Set the number lanes to be used on the DisplayPort link. Please note that writing this CI has no immediate effect. The configured value is used the next time there is a link training, or when the command to apply setting without LT is issued. Valid settings are 1, 2 and 4 lanes.

5.15.2 TSI_DPTX_LINK_CFG_BIT_RATE

```
TSI_DPTX_LINK_CFG_BIT_RATE                0x781
(TSI_SRC_DP_LINK_CFG_BIT_RATE             0x781)
unsigned int dp_src_bit_rate              U32
4 bytes                                   RW
```

Synopsis

Defines the bit rate as multiplier of 0.27Gbps. Please note that writing this CI has no immediate effect. The configured value is used the next time there is a link training, or when the command to apply setting without LT is issued. Valid settings are 6, 10, 20 and 30.

5.15.3 TSI_DPTX_LINK_CFG_FLAGS

```
TSI_DPTX_CFG_FLAGS                                  0x782
(TSI_SRC_DP_CFG_FLAGS                               0x782)
unsigned int dp_src_framing                         U32
4 bytes                                             RW
```

Synopsis

Select misc. features for the DP link. Please note that writing this CI has no immediate effect. The configured value is used the next time there is a link training. See table below for defined flags:

Bits	Description	
0	RESERVED	
1	TPS3	
	0	Disable TPS3
	1	Enable TPS3
2	Framing selection	
	0	Standard framing
	1	Enhanced framing
31:3	RESERVED	

5.15.4 TSI_DPTX_OVERRIDE_VOLTAGE_SWING

```
TSI_DPTX_OVERRIDE_VOLTAGE_SWING                0x783
(TSI_SRC_DP_OVERRIDE_VOLTAGE_SWING             0x783)
unsigned int dp_src_ovr_voltage_swing          U32
4 bytes                                        RW
```

Synopsis

Writing this CI will override the voltage swing values. If no override values have been written reading the CI will fail. See table below for definitions:

Bits	Descirption	
7:0	Voltage swing for Lane 0	
	0	400 mVpp
	1	600 mVpp
	2	800 mVpp
	3	1200 mVpp
	*	RESERVED
15:8	Voltage swing for Lane 1. The bit-field values are the same as with Lane 0 (See above).	
23:16	Voltage swing for Lane 2. The bit-field values are the same as with Lane 0 (See above).	
31:24	Voltage swing for Lane 3. The bit-field values are the same as with Lane 0 (See above).	

5.15.5 TSI_DPTX_OVERRIDE_PRE_EMPHASIS

```
TSI_DPTX_OVERRIDE_PRE_EMPHASIS                 0x784
(TSI_SRC_DP_OVERRIDE_PRE_EMPHASIS              0x784)
unsigned int dp_src_ovr_pre_emphasis           U32
4 bytes                                        RW
```

Synopsis

Writing this CI will override the pre emphasis values. Reading the CI will return the previous values written. See table below for definitions:

Bits	Description	
7:0	Pre-emphasis setting for Lane 0.	
	0	0 dB pre-emphasis
	1	3.5 dB pre-emphasis
	2	6 dB pre-emphasis
	3	9.5 dB pre-emphasis
	*	RESERVED
15:8	Pre-emphasis setting for Lane 1. The bit-field values are the same as with Lane 0 (See above).	
23:16	Pre-emphasis setting for Lane 2. The bit-field values are the same as with Lane 0 (See above).	
31:24	Pre-emphasis setting for Lane 3. The bit-field values are the same as with Lane 0 (See above).	

5.15.6 TSI_DPTX_LINK_PATTERN

```
TSI_DPTX_LINK_PATTERN                          0x785
(TSI_SRC_DP_LINK_PATTERN                       0x785)
unsigned int dp_src_lt_pattern                 U32
4 bytes                                        RW
```

Synopsis

Force device to output a pattern that is typically used only with Link Training. See table below for options:

ID	Description
0	Active video
1	Idle pattern
2	Training pattern 1
3	Training pattern 2
4	Training pattern 3
5	Training pattern 4
6	PRBS7
7	HBR2
8	SER

5.15.7 TSI_W_DPTX_COMMAND

```
TSI_W_DPTX_COMMAND                             0x786
(TSI_W_SRC_DP_COMMAND                          0x786)
unsigned int dp_src_cmd                        U32
4 bytes                                        WO
```

Synopsis

Carry out commands on the source. See table below for available commands:

ID	Description
0	No operation. Writing this has no effect.
1	Begin link-training.
2	Apply bit-rate and lane count settings without performing link-training.
3	Enable MST mode (Please refer to 5.18.1 TSI_PG_ENABLED_STREAM_COUNT to enable streams)
4	Disable MST mode

5.15.8 TSI_W_DPTX_DPCD_BASE

```
TSI_W_DPTX_DPCD_BASE                          0x793
unsigned int dprx_dpcd_base_ptr               U32
4 bytes                                       WO
```

Description

DPCD read and write start pointer into the DPCD register space. The DPCD address value may not exceed 0x00FFFFFF. Default value is 0.

5.15.9 TSI_DPTX_DPCD_DATA

```
TSI_DPTX_DPCD_DATA                            0x794
unsigned char dprx_dpcd_data[]                ARRAY_U8
Variable size                                 RW
```

Description

Read and/or write DPCD registers. Each DPCD register is one byte (8 bits). Read/Write access size is not limited, but access size + DPCD base address may not exceed 0x01000000. Please refer to DP Specifications in order to decode the DPCD register data.

5.16 DP Source – Link status

ClientVersion 11, and higher *<license:TBD>*

5.16.1 TSI_R_DPTX_HPD_STATUS

```
TSI_R_DPTX_HPD_STATUS                          0x787
(TSI_R_SRC_DP_HPD_STATUS                       0x787)
unsigned int dp_src_HPD_status                 U32
4 bytes                                        RO
```

Synopsis

Used to read HPD signal logical status. (0 = HPD de-asserted, 1 = HPD asserted).

5.16.2 TSI_R_DPTX_LT_RESULT

```
TSI_R_DPTX_LT_RESULT                           0x788
(TSI_R_SRC_DP_LT_RESULT                        0x788)
unsigned int dp_src_lt_res                     U32
4 bytes                                        RO
```

Synopsis

Contains result of previous link training. See table below for values:

Bits	Description	
7:0	Link training procedure result	
	0	Link training not started
	1	Link training in progress
	2	Link training failed.
	3	Link training succeeded.
	*	RESERVED
31:8	RESERVED	

5.16.3 TSI_R_DPTX_LINK_STATUS_BITS

```
TSI_R_DPTX_LINK_STATUS_BITS                    0x789
(TSI_R_SRC_DP_LINK_STATUS_BITS                 0x789)
unsigned int dp_src_lt_status                  U32
4 bytes                                        RO
```

Synopsis

Indicates Clock Recovery, Channel equalization and symbol locks states for each lane. The status is the result of the previous link training. See table below for bit definitions:

Bits	Description
0	Clock Recovery done for Lane 0.
1	Channel EQ done for Lane 0.
2	Symbol lock for Lane 0.
3	RESERVED
4	Clock Recovery done for Lane 1.
5	Channel EQ done for Lane 1.
6	Symbol lock for Lane 1.
7	RESERVED
8	Clock Recovery done for Lane 2.
9	Channel EQ done for Lane 2.
10	Symbol lock for Lane 2.
11	RESERVED
12	Clock Recovery done for Lane 3.
13	Channel EQ done for Lane 3.
14	Symbol lock for Lane 3.
30:15	RESERVED
29	MST Support flag. (1 = MST Supported)
30	MST / SST Status. (1 = MST, 0 = SST)
31	ILA

5.16.4 TSI_R_DPTX_LINK_STATUS_VOLT_SWING

```
TSI_R_DPTX_LINK_STATUS_VOLT_SWING                0x78a
(TSI_R_SRC_DP_LINK_STATUS_VOLT_SWING             0x78a)
unsigned int dp_src_lt_voltage_swing             U32
4 bytes                                          RO
```

Synopsis

Indicates voltage swing values for each lane. The state is the result of the previous link training. See table below for definitions:

Bits	Descirption	
7:0	Voltage swing for Lane 0	
	0	400 mVpp
	1	600 mVpp
	2	800 mVpp
	3	1200 mVpp
	*	RESERVED
15:8	Voltage swing for Lane 1. The bit-field values are the same as with Lane 0 (See above).	
23:16	Voltage swing for Lane 2. The bit-field values are the same as with Lane 0 (See above).	
31:24	Voltage swing for Lane 3. The bit-field values are the same as with Lane 0 (See above).	

5.16.5 TSI_R_DPTX_LINK_STATUS_LANE_COUNT

```
TSI_R_DPTX_LINK_STATUS_LANE_COUNT                0x78b
(TSI_R_SRC_DP_LINK_STATUS_LANE_COUNT             0x78b)
unsigned int dp_src_lane_count                   U32
4 bytes                                          RO
```

Synopsis

Indicates number of lanes achieved in the previous link training.

5.16.6 TSI_DPTX_LINK_STATUS_BIT_RATE

```
TSI_R_DPTX_LINK_STATUS_BIT_RATE                0x78c
(TSI_R_SRC_DP_LINK_STATUS_BIT_RATE             0x78c)
unsigned int dp_src_bit_rate                   U32
4 bytes                                        RO
```

Synopsis

Indicates link bit-rate achieved during the previous link training as multiple of 0.27Gbps.

5.16.7 TSI_R_DPTX_LINK_STATUS_PRE_EMP

```
TSI_R_DPTX_LINK_STATUS_PRE_EMP                 0x78d
(TSI_R_SRC_DP_LINK_STATUS_PRE_EMP              0x78d)
unsigned int dp_src_pre_emp                    U32
4 bytes                                        RO
```

Synopsis

Indicates the pre-emphasis setting achieved during the previous link training. See table below for definitions:

Bits	Description	
7:0	Pre-emphasis setting for Lane 0.	
	0	0 dB pre-emphasis
	1	3.5 dB pre-emphasis
	2	6 dB pre-emphasis
	3	9.5 dB pre-emphasis
	*	RESERVED
15:8	Pre-emphasis setting for Lane 1. The bit-field values are the same as with Lane 0 (See above).	
23:16	Pre-emphasis setting for Lane 2. The bit-field values are the same as with Lane 0 (See above).	
31:24	Pre-emphasis setting for Lane 3. The bit-field values are the same as with Lane 0 (See above).	

5.16.8 TSI_R_DPTX_CRC_R

```
TSI_R_DPTX_CRC_R                               0x790
unsigned int dptx_crc_red                      U32
4 bytes                                        R
```

Description

Get red CRC value for current generated video stream.

5.16.9 TSI_R_DPTX_CRC_G

TSI_R_DPTX_CRC_G	0x791
unsigned int dptx_crc_green	U32
4 bytes	R

Description

Get green CRC value for current generated video stream.

5.16.10 TSI_R_DPTX_CRC_B

TSI_R_DPTX_CRC_B	0x792
unsigned int dptx_crc_blue	U32
4 bytes	R

Description

Get blue CRC value for current generated video stream.

5.17 Configuration items for USB Type-C

ClientVersion 9, and higher *Advanced License required*

This section defines the new configuration items that are specific to UCD-340 and USB Type-C. The CI Space from 0x600 to 0x6ff is reserved for USB-Type C specific controls.

Name	Description	Reference
TSI_W_USBC_CABLE_CONTROL	Cable related controls	5.17.1
TSI_W_USBC_INITIAL_ROLE	USB Type-C role control	5.17.2
TSI_USBC_DP_ALT_MODE_SETUP	DP Alternate mode settings	5.17.3
TSI_W_USBC_ROLE_CONTROL	USB Type-C Role swap requests	5.17.4
TSI_W_USBC_ROLE_CONTROL_SWAP	USB Type-C Enable/Disble SWAP requests.	1.17.5
TSI_W_USBC_DP_ALT_MODE_COMMAND	DP Alternate mode commands	5.17.6
TSI_R_USBC_TE_HW_CONFIGURATION	HW Configuration info	5.17.7
TSI_R_USBC_CABLE_STATUS	Cable status information	5.17.8
TSI_R_USBC_ROLE_STATUS	USB Type-C Role status information	5.17.10
TSI_R_USBC_ROLE_CONTROL_SWAP	USB Type-C SWAP messaging status	1.17.11
TSI_R_USBC_DP_ALT_MODE_STATUS	DP Alternate mode status information	5.17.12
TSI_R_USBC_POWER_STATUS	USB Type-C Power status information	5.17.13
TSI_R_USBC_POWER_SOURCE_PDO	Power contract information	5.17.14
TSI_R_USBC_POWER_SINK_RDO	Power contract information	5.17.15
TSI_R_USBC_IDO_TABLE	Cable E-marker information	5.17.9
TSI_USBC_PWR_COMMAND	Apply pending power data objects	5.17.26
TSI_USBC_PWR_LOCAL_SOURCE_PDO_SELECT	Select source PDO for editing	5.17.27
TSI_USBC_PWR_LOCAL_SOURCE_PDO_TYPE	Set source PDO type	5.17.28
TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_CURRENT	Set source PDO maximum current	5.17.29
TSI_USBC_PWR_LOCAL_SOURCE_PDO_VOLTAGE	Set source PDO voltage	5.17.30
TSI_USBC_PWR_LOCAL_SOURCE_PDO_PEAK_CURRENT	Set source PDO peak current	5.17.31
TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_POWER	Set source PDO maximum power	5.17.32
TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_VOLTAGE	Set source PDO maximum voltage	5.17.33
TSI_USBC_PWR_LOCAL_SOURCE_PDO_MIN_VOLTAGE	Set source PDO minimum voltage	5.17.34
TSI_USBC_PWR_LOCAL_SOURCE_PDO_FIXED_SUPPLY_BITS_25_TO_29	Set source PDO bits 25 through 29	5.17.35
TSI_USBC_PWR_LOCAL_SINK_PDO_SELECT	Select sink PDO for editing	5.17.36
TSI_USBC_PWR_LOCAL_SINK_PDO_TYPE	Set sink PDO type	5.17.37
TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_CURRENT	Set sink PDO maximum current	5.17.38
TSI_USBC_PWR_LOCAL_SOURCE_PDO_VOLTAGE	Set sink PDO voltage	5.17.39
TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_POWER	Set sink PDO maximum power	5.17.40
TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_VOLTAGE	Set sink PDO maximum voltage	5.17.41
TSI_USBC_PWR_LOCAL_SINK_PDO_MIN_VOLTAGE	Set sink PDO minimum voltage	5.17.42
TSI_USBC_PWR_LOCAL_SINK_PDO_FIXED_SUPPLY_BITS_25_TO_29	Set sink PDO bits 25 through 29	5.17.43

5.17.1 TSI_W_USBC_CABLE_CONTROL

```
TSI_W_USBC_CABLE_CONTROL                    0x600
unsigned int USBC_cable_control             U32
4 bytes                                     WO
```

Synopsis

Command CI Used to control cable related features, like connection, orientation and power sourcing capability.

Command ID	Description
0	No operation. Writing zero has no effect.
1	Set cable to “straight” orientation. **This is default orientation.** ***Important:*** *using this command requires the Unigraf Electrical Testing cable to be connected to the USB-Type C port.*
2	Set cable to “Flipped” orientation. ***Important:*** *using this command requires the Unigraf Electrical Testing cable to be connected to the USB-Type C port.*
3	Indicate 0.9A power source capability.
4	Indicate 1.5A power source capability.
5	Indicate 3.0A power source capability. **This is default pull-up selection**
6	Disconnect CC lines. The DUT will see this as cable unplugged. **Default after device open.**
7	Connect CC lines. The DUT will see this as cable plugged.
8	Set behavior for “One differential pair for USB 2.0” electrical test cable type.
9	Set behavior for “Two differential pairs for USB 2.0” electrical test cable type.
10	Disable USB 2.0 PHY.
11	Disable USB 2.0 PHY and charger.
12	Enable USB 2.0 PHY.
13	Enable USB 2.0 PHY and charger; Device mode.
14	Enable USB 2.0 PHY; Host mode. (Charger disabled).
15	Disable USB 3.0 PHY.
16	Enable USB 3.0 PHY.

5.17.2 TSI_W_USBC_INITIAL_ROLE

```
TSI_W_USBC_INITIAL_ROLE                          0x601
unsigned int USBC_initial_role                   U32
4 bytes                                          WO
```

Synopsis

Command CI Used to control initial role settings and related configuration. Issue any commands here before issuing the "Connect CC lines" command on the 5.17.1 TSI_W_USBC_CABLE_CONTROL.

Command ID	Description
0	No operation. Writing zero has no effect.
1	Set initial port role to UFP.
2	Set initial port role to DFP.
3	Set initial port role to DRP. ***Important:*** *Actual mode after cable plug will not be DRP – Instead it will be either UFP or DFP depending on connected DUT. If connected DUT is also DRP, it is not possible to predict which role will be selected by TE due to the way that DRP is implemented in the USB-Type C specifications.*

5.17.3 TSI_USBC_DP_ALT_MODE_SETUP

```
TSI_USBC_DP_ALT_MODE_SETUP                    0x602
unsigned int USBC_DP_alt_mode_setup           U32
4 bytes                                       RW
```

Synopsis

DisplayPort alternate mode capabilities and feature definitions.

Bit	Description	
1:0	RESERVED	
4:2	Pin mapping capability flags for DisplayPort Sink alt.mode.	
	Bit	
	0	Support for pin mapping “C”: DP v1.3, 4 lanes
	1	Support for pin mapping “D”: DP v1.3, 2 lanes + USB 3.1
	2	Support for pin mapping “E”: DP v1.3, 4 lanes
9:5	RESERVED	
12:10	Pin mapping capability flags for DisplayPort Source alt.mode.	
	Bit	
	0	Support for pin mapping “C”: DP v1.3, 4 lanes
	1	Support for pin mapping “D”: DP v1.3, 2 lanes + USB 3.1
	2	Support for pin mapping “E”: DP v1.3, 4 lanes
29:13	RESERVED	
30	Multi-function preference setting	
	0	Don’t prefer multi-function connections.
	1	Prefer multi-function connections.
31	DP Alternate mode auto enter on cable plug	
	0	Auto-enter functionality is disabled
	1	Auto-enter functionality is enabled

5.17.4 TSI_W_USBC_ROLE_CONTROL

```
TSI_W_USBC_ROLE_CONTROL                          0x603
unsigned int USBC_role_control                   U32
4 bytes                                          WO
```

Synopsis

Command CI to control port roles after cable is plugged into TE and roles are established. Once the command is issued, please read the 5.17.10 TSI_R_USBC_ROLE_STATUS CI to check if the role was actually changed.

Important: *A role swap is an expensive operation that can take up to 5 seconds to complete depending on used DUT.*

Important: *The TE can only request swapping of the roles, DUT can always reject the request leading to no change in the respective role.*

Command ID	Description
0	No operation. Writing zero has no effect.
1	Request swapping data role.
2	Request swapping power role.
3	Request swapping Vconn.
4	Clear “USB Communications capable” flag for PD Source
5	Set “USB Communications capable” flag for PD Source
6	Clear “USB Communications capable” flag for PD Sink
7	Set “USB Communications capable” flag for PD Sink

5.17.5 TSI_W_USBC_ROLE_CONTROL_SWAP

```
TSI_W_USBC_ROLE_CONTROL_SWAP                    0x619
unsigned int USBC_role_control_swap             U32
4 bytes                                         WO
```

Synopsis

.Command CI to enable or disable swap message processing.

Command ID	Description
0	Disable Vconn_Swap message processing.
1	Enable Vconn_Swap message processing.
2	Disable PR_Swap message processing.
3	Enable PR_Swap message processing.
4	Disable DR_Swap message processing.
5	Enable DR_Swap message processing.

5.17.6 TSI_W_USBC_DP_ALT_MODE_COMMAND

```
TSI_W_USBC_DP_ALT_MODE_COMMAND                0x604
unsigned int USBC_DP_alt_mode_command         U32
4 bytes                                       WO
```

Synopsis

Used to manually control USB Alternate mode.

Important: *Issuing any command (except for "No operation") will clear the "DP alternate mode auto enter on cable plug" bit in 5.17.3 TSI_USBC_DP_ALT_MODE_SETUP CI.*

Command ID	Description
0	No operation. Writing zero has no effect.
1	Exit any DP Alternate mode currently in effect. ***Important:*** *This command is only available in the DFP role.*
2	Ignore any future DP Alternate mode entry requests originating from the DUT. ***Important:*** *This command is only available in the DFP role.*
3	Enter DP Alternate mode with pin assignment "D": DP v1.3, 2 lanes + USB 3.1. ***Important:*** *This command is only available in the DFP role.*
4	Enter DP Alternate mode with pin assignment "C": DP v1.3, 4 lanes. ***Important:*** *This command is only available in the DFP role.*

5.17.7 TSI_R_USBC_TE_HW_CONFIGURATION

```
TSI_R_USBC_TE_HW_CONFIGURATION                0x605
unsigned int USBC_TE_HW_config                U32
4 bytes                                       RO
```

Synopsis

Contains information about the optional hardware modules and external devices.

Bit	Description
0	Unigraf External power-supply / Power-sink device is connected to the TE.
1	Electrical Testing board is installed on the TE.
31:3	RESERVED

5.17.8 TSI_R_USBC_CABLE_STATUS

```
TSI_R_USBC_CABLE_STATUS                          0x606
unsigned int USBC_cable_status                   U32
4 bytes                                          RO
```

Synopsis

Contains cable-related status data.

Bit		
0	Cable plugged state	
	0	Cable is unplugged
	1	Cable is plugged
1	Cable orientation	
	0	Straight orientation
	1	Flipped orientation
2	Indicates the connection status	
	0	Disconnected
	1	Connected
3	Cable E-Marking ***Important:*** *This information is valid only in DFP data role.*	
	0	Unmarked cable is attached.
	1	E-Marked cable is attached.
4	Unigraf Electrical testing cable detect	
	0	Normal USB-C cable
	1	Unigraf Electrical Testing cable is attached.
5	Indication for E-Marker IDO table in 5.17.9 TSI_R_USBC_IDO_TABLE validity. ***Important:*** *It may take a while to read the IDO table from an E-marked cable. The E-Marker should become available if the data role is DFP and the cable is E-Marked.*	
	0	E-Marker IDO table does not contain valid information.
	1	E-Marker IDO table contains valid information.
6	USB 2.0 PHY State	
	0	USB 2.0 PHY is disabled.
	1	USB 2.0 PHY is enabled.
7	USB 3.0 PHY State	
	0	USB 3.0 PHY is disabled.
	1	USB 3.0 PHY is enabled.
31:5	RESERVED	

5.17.9 TSI_R_USBC_IDO_TABLE

```
TSI_R_USBC_IDO_TABLE                          0x60c
unsigned int USBC_IDO_Table[]                 ARRAY_U32
Variable size, Max. 24 bytes                  RO
```

Synopsis

Provides access to Identity Data Objects (IDO) which are received from near cable plug as the reply to Discover Identity request. The content of this table is cleared on reset or cable unplug event.

Index	IDO	Description
0	ID Header	See 6.4.4.3.1.1 of PD specification
1	Cert stat	32-bit integer assigned by USB-IF
2	Product	See 6.4.4.3.1.9 of PD specification
3..5	Produc type specific	See 6.4.4.3.1.10.1 for passive cable VDO, 6.4.4.3.1.10.2 for active cable VDO.

5.17.10 TSI_R_USBC_ROLE_STATUS

```
TSI_R_USBC_ROLE_STATUS                        0x607
unsigned int USBC_role_status                 U32
4 bytes                                       RO
```

Synopsis

Indicates the current device roles.

Bit		
0	Data Role	
	0	Up Facing Port (UFP)
	1	Down Facing Port (DFP)
1	Power role	
	0	Source
	1	Sink
2	Vconn	
	0	Off
	1	On
31:3	RESERVED	

5.17.11 TSI_R_USBC_ROLE_CONTROL_SWAP

```
TSI_R_USBC_ROLE_CONTROL_SWAP                  0x620
unsigned int USBC_role_swap                   U32
4 bytes                                       RO
```

Synopsis

Command CI to query if swap message processing is enabled or disabled.

Bit	Description	
0	Vconn	
	0	Vconn_Swap message processing is disabled.
	1	Vconn_Swap message processing is enabled.
1	Power role	
	0	PR_Swap message processing is disabled.
	1	PR_Swap message processing is enabled.
2	Data Role	
	0	DR_Swap message processing is disabled.
	1	DR_Swap message processing is enabled.
31:3	RESERVED	

5.17.12 TSI_R_USBC_DP_ALT_MODE_STATUS

```
TSI_R_USBC_DP_ALT_MODE_STATUS                    0x608
unsigned int USBC_alt_mode                       U32
4 bytes                                          RO
```

Synopsis

Indicates the current alternate mode and it's setup

Bit	Description	
0	DisplayPort Alternate mode mode indicator	
	0	DisplayPort alternate mode is not in use ***Important:*** *In this mode, the 2:1, 6:3 and 11:8 bit-fields do not contain valid data!*
	1	DisplayPort alternate mode is active.
2:1	DisplayPort Alternate mode configuration	
	0	Set configuration USB
	1	Set UFP_U as DFP_D
	2	Set UFGP_U as UFP_D
	3	RESERVED
6:3	Signaling	
	0	Unspecified
	1	DP v1.3
	2	GEN2
	*	RESERVED
7	RESERVED	
11:8	Pin Assignment	
	0	No pin assignment
	1	“A”: DisplayPort GEN2 4/2 lanes
	2	“B”: DisplayPort GEN2 2/1 lanes
	3	“C”: DisplayPort v1.3 4 lanes (USB Type-C cable)
	4	“D”: DisplayPort v1.3 2 lanes + USB 3.1 (USB Type-C cable)
	5	“E”: DisplayPort v1.3 4 lanes (USB Type-C to DP adapter)
	6	“F”: DisplayPort v1.3 2 lanes + USB GEN1 (USB Type-C to DP adapter
	*	RESERVED
31:12	RESERVED	

5.17.13 TSI_R_USBC_POWER_STATUS

```
TSI_R_USBC_POWER_STATUS                          0x609
unsigned int USBC_power_status                   U32
4 bytes                                          RO
```

Synopsis

Indicates current power status.

Important: *The indicated internal loading resistor values (bits 8:5) apply to UCD-340 front-end REV-C only.*

Bit	Description	
1:0	Power sink current	
	0	Legacy current selected: 0.9A current.
	1	Legacy current selected: 1.5A current
	2	Legacy current selected: 3.0A current
	3	Defined by power contract
2	Power contract established	
	0	Power contract not established.
	1	Power contract established.
4:3	RESERVED	
8:5	Internal loading resistors for power sinking.	
	0	Internal load resistor connections are open: no internal loading is applied.
	1	10Ω (0.5A) resistive load applied. (This setting is used for 0.5A USB power loading)
	2	5.6Ω (0.89A) resistive load applied. (This setting is used for 0.9A USB power loading)
	3	3.58Ω (1.39A) resistive load applied.
	4	3.4Ω (1.47A) resistive load applied. (This setting is used for 1.5A USB power loading)
	5	2.54Ω (1.97A) resistive load applied.
	6	2.12Ω (2.36A) resistive load applied.
	7	1.74Ω (2.86A) resistive load applied. (This setting is used for 3.0A USB power loading)
	*	RESERVED
31:9	RESERVED	

5.17.14 TSI_R_USBC_POWER_SOURCE_PDO

```
TSI_R_USBC_POWER_SOURCE_PDO                    0x60a
unsigned int USBC_power_pdo                    U32
4 bytes                                        RO
```

Synopsis

This CI provides access to the PDO that was requested by a power sink device from the TE. Data in this CI is valid if the power role is Source, and power contract established bit (2) is set in 5.17.13 TSI_R_USBC_POWER_STATUS CI.

The PDO Data is in raw form as defined in USB DP Standard. Also see TSI_USBC_PWR_LOCAL_SOURCE_PDO.

5.17.15 TSI_R_USBC_POWER_SINK_RDO

```
TSI_R_USBC_POWER_SINK_RDO                      0x60b
unsigned int USBC_power_rdo                    U32
4 bytes                                        RO
```

Synopsis

This CI provides access to the RDO selected by TE. Data in this CI is valid if the power role is Sink, and power contract established bit (2) is set in 5.17.13 TSI_R_USBC_POWER_STATUS CI.

The RDO Data in RAW form as defined in USB DP Standard.

Fixed and Variable RDO:

Bits	Parameters
31	Reserved – Shall be set to zero
30:28	Object position (1-7)
27	GiveBack flag
26	Capability mismatch
25	USB communications capable
24	No USB suspend
23	Unchunked Extended Messages Supported
22:20	Reserved – Shall be set to zero
19:10	Operating current in 10-mA units
9:0	Maximum operating current in 10-mA units

Battery RDO:

Bits	Parameters
31:30	Reserved – Shall be set to zero
30:28	Object position (1-7)
27	GiveBack flag = 0
26	Capability mismatch
25	USB communications capable

24	No USB suspend
23	Unchunked Extended Messages Supported
22:20	Reserved – Shall be set to zero
19:10	Operating power in 250-mW units
9:0	Maximum operating power in 250-mW units

5.17.16 TSI_R_USBC_IDO_TABLE

```
TSI_R_USBC_IDO_TABLE                          0x60c
unsigned int USBC_IDO_Table[]                 ARRAY_U32
Variable size                                 RO
```

Synopsis

This CI provides access to USB-C IDO table. Read table with zero size to get required buffer size, or use sufficient buffer size on first attempt.

5.17.17 TSI_USBC_EPU_LOAD_CONTROL

```
TSI_USBC_EPU_LOAD_CONTROL                    0x60d
unsigned int EPU_Load_Bits                   U32
4 bytes                                      RW
```

Synopsis

Note that newer and more flexible configuration items have been added to set and get external and internal loads: TSI_USBC_RESISTANCE_CTRL, TSI_R_USBC_INT_RESISTANCE_STATUS, TSI_R_USBC_EXT_RESISTANCE_STATUS.

TSI_USBC_EPU_LOAD_CONTROL provides access to control loading resistors on external power unit. See below for valid selections:

Value	Description
0x001	Select EPU loading resistor configuration for 7.6 Ω load
0x002	Select EPU loading resistor configuration for 6.6 Ω load
0x004	Select EPU loading resistor configuration for 5.6 Ω load
0x008	Select EPU loading resistor configuration for 4.6 Ω load
0x010	Select EPU loading resistor configuration for 3.6 Ω load
0x020	Select EPU loading resistor configuration for 13.9 Ω load
0x080	Select EPU loading resistor configuration for 1.8 Ω load
0x100	Select EPU loading resistor configuration for 9.1 Ω load
0x200	Select EPU loading resistor configuration for 10.6 Ω load
*	RESERVED – Do not use!

5.17.18 TSI_USBC_PWR_CONTRACT_CONTROL

```
TSI_USBC_PWR_CONTRAC_CONTROL                    0x60e
unsigned int PWR_Contract_Control               U32
4 bytes                                         RW
```

Synopsis

Defines how the UCD-340 selects which of the offered power contract options is to be selected. See table below for configuration options:

Bit	Description	
0	Auto negotiate flag	
	0	Don't auto-negotiate
	1	Automatically negotiate power contract, use fixed voltage PDO
1	Use battery PDO	
	0	Don't use battery PDO for power contract
	1	Use batter PDO for power contract negotiation.
2	Use variable PDO	
	0	Don't use variable power PDO for power contract
	1	Use variable power PDO in power contract negotiation.
3	USB Communication capable flag (copied from 5.17.15 TSI_R_USBC_POWER_SINK_RDO, bit #25).	
5:4	Contract preference	
	0	Prefer higher current power contract
	1	Prefer higher voltage power contract
	2	Prefer higher power power contract
	3	RESERVED
6	No USB suspend flag (copied from 5.17.15 TSI_R_USBC_POWER_SINK_RDO, bit #24).	
7	"Give back flag" (copied from 5.17.15 TSI_R_USBC_POWER_SINK_RDO, bit #27).	
8	Automatic minimum power	
	0	Automatically calculate minimum required power.
	1	Don't calculate minimum power.
25:16	Minimum required power.	
31	Manual power contract selection.	
	0	Auto-negotiate power contract.
	1	Power contract selected by index CI. (5.17.19 TSI_USBC_PWR_CONTRACT_SELECT)

5.17.19 TSI_USBC_PWR_CONTRACT_SELECT

```
TSI_USBC_PWR_CONTRACT_SELECT                    0x60f
unsigned int PWR_Contract_Sel                   U32
4 bytes                                         RW
```

Synopsis

If bit 31 is set in 5.17.18 TSI_USBC_PWR_CONTRACT_CONTROL, this CI is used to select which PDO is used to establish the power contract with link partner.

5.17.20 TSI_USBC_PWR_LOCAL_SINK_PDO

```
TSI_USBC_PWR_LOCAL_SINK_PDO                     0x610
unsigned int LocalSinkPDO[]                     ARRAY_U32
Variable size                                   RW
```

Synopsis

Contains binary images of PDO's that advertise the TE power sinking capabilities. Maximum number of PDO's is 7. For details on the format of these PDO's, please refer to "USB DP standard" (Tables 6-6, 6-8, 6-9).

The type is determined via bits 31:30 as shown below:

Bits	Value	Type
31:30	0	Fixed supply
	1	Battery
	2	Variable supply
	3	Reserved

Fixed supply the PDO sink:

Bits	Parameters
31:30	Fixed supply (0)
29	Dual-role power
28	Higher capability
27	Externally powered
26	USB communications capable
25	Dual-role data
24:23	Fast Role Swap required USB Type-C Current
22:20	Reserved – Shall be set to zero
19:10	Voltage in 50-mV units
9:0	Operational current in 10-mA units

Variable-Supply (Nonbattery) PDO Sink

Bits	Parameters
31:30	Variable supply (2)
29:20	Maximum voltage in 50-mV units
19:10	Minimum voltage in 50-mV units
9:0	Maximum current in 10-mA units

Battery-Supply PDO Sink

Bits	Parameters
31:30	Battery supply (1)
29:20	Maximum voltage in 50-mV units
19:10	Minimum voltage in 50-mV units
9:0	Maximum allowable power in 250-mW units

5.17.21 TSI_USBC_PWR_LOCAL_SOURCE_PDO

```
TSI_USBC_PWR_LOCAL_SOURCE_PDO                    0x611
unsigned int LocalSourcePDO[]                    ARRAY_U32
Variable size                                    RW
```

Synopsis

Contains binary images of PDO's that advertise the TE power sourcing capabilities. Maximum number of PDO's is 7. For details on the format of these PDO's, please refer to "USB DP standard" (Tables 6-6, 6-8, 6-9).

The type is determined via bits 31:30 as shown below:

Bits	Value	Type
31:30	0	Fixed supply
	1	Battery
	2	Variable supply
	3	Reserved

Fixed supply the PDO source

Bits	Parameters
31:30	Fixed supply (0)
29	Dual-role power
28	USB suspend supported
27	Externally powered
26	USB communications capable
25	Dual-role data
24	Unchunked Extended Messages Supported
23:22	Reserved – Shall be set to zero
21:20	Peak current
19:10	Voltage in 50-mV units
9:0	Maximum current in 10-mA units

Variable-Supply (Nonbattery) PDO Source

Bits	Parameters
31:30	Variable supply (2)
29:20	Maximum voltage in 50-mV units
19:10	Minimum voltage in 50-mV units
9:0	Maximum current in 10-mA units

Battery-Supply PDO Source

Bits	Parameters
31:30	Battery supply (1)
29:20	Maximum voltage in 50-mV units
19:10	Minimum voltage in 50-mV units
9:0	Maximum allowable power in 250-mW units

5.17.22 TSI_R_USBC_PWR_REMOTE_SINK_PDO

```
TSI_R_USBC_PWR_REMOTE_SINK_PDO                0x612
unsigned int RemoteSinkPDO[]                  ARRAY_U32
Variable size                                 RO
```

Synopsis

Contains binary images of PDO's received from DUT advertising the DUT's power sink requirements. Maximum number of PDO's is 7. For details on the format of these PDO's, please refer to "USB DP standard" (Tables 6-6, 6-8, 6-9). Also see TSI_USBC_PWR_LOCAL_SINK_PDO.

5.17.23 TSI_R_USBC_PWR_REMOTE_SOURCE_PDO

```
TSI_R_USBC_PWR_REMOTE_SOURCE_PDO                0x613
unsigned int RemoteSourcePDO[]                  ARRAY_U32
Variable size                                   RO
```

Synopsis

Contains binary images of PDO's received from DUT advertising the DUT's power sourcing capabilities. Maximum number of PDO's is 7. For details on the format of these PDO's, please refer to "USB DP standard" (Tables 6-6, 6-8, 6-9). Also see TSI_USBC_PWR_LOCAL_SOURCE_PDO.

5.17.24 TSI_R_USBC_PD_STATUS

```
TSI_R_USBC_PD_STATUS                            0x614
unsigned int PD_status                          U32
4 bytes                                         RO
```

Synopsis

Indicates the status of the PD controller in USB-C enabled devices. If the value is zero, the PD controller is operating normally. If the value is 1, the PD controller is in a failure state and needs to be reset – In this case, please use TSI_W_USBC_PD_COMMAND CI to issue PD Reset command. Other values are reserved for future additions and should be ignored.

5.17.25 TSI_W_USBC_PD_COMMAND

```
TSI_W_USBC_PD_COMMAND                           0x615
unsigned int PD_command                         U32
4 bytes                                         WO
```

Synopsis

Used to issue commands to the PD Controller in USB-C enabled devices. The table below lists the available commands:

Command	Description
0	No operation.
1	Reset PD Controller. This command should be used if the PD_STATUS is indicated as "Failed" (=1).
<Other>	RESERVED.

5.17.26 TSI_USBC_PWR_COMMAND

```
TSI_USBC_PWR_COMMAND                          0xfe0
unsigned int PWR_command                      U32
4 bytes                                       WO
```

Synopsis

Used to write pending Power Data Objects (PDO's) in USB-C enabled devices. Pending PDO's replace only those that are pending. Before this command can be applied, the PDO must (in this order) be selected, its type assigned, and finally the relevant parameters assigned (See CI's below). For source PDO data structures see TSI_USBC_PWR_LOCAL_SOURCE_PDO. For sink PDO data structures see TSI_USBC_PWR_LOCAL_SINK_PDO.The table below lists the available commands:

Command	Description
1	Apply pending local source PDO's.
2	Apply pending local sink PDO's.
<Other>	RESERVED.

5.17.27 TSI_USBC_PWR_LOCAL_SOURCE_PDO_SELECT

```
TSI_USBC_PWR_LOCAL_SOURCE_PDO_SELECT          0xfe1
unsigned int Source_PDO_select                U32
4 bytes                                       RW
```

Synopsis

Used to select the source Power Data Object (PDO) by number (0-6) to be added or edited in USB-C enabled devices. This is the first step when we wish to add or modify a source PDO.

5.17.28 TSI_USBC_PWR_LOCAL_SOURCE_PDO_TYPE

```
TSI_USBC_PWR_LOCAL_SOURCE_PDO_TYPE            0xfe2
unsigned int Source_PDO_type                  U32
4 bytes                                       RW
```

Synopsis

Used to assign the Power Data Object (PDO) type (0-3) to the selected source PDO (via TSI_USBC_PWR_LOCAL_SOURCE_PDO_SELECT CI above). Values are: 0 to disable, 1 for fixed power supply, 2 for variable power suppy and 3 for battery. This is the second step when we wish to add or modify a PDO. An error will occur if no PDO source object is selected.

5.17.29 TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_CURRENT

```
TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_CURRENT     0xfe3
unsigned int Source_PDO_max_current           U32
4 bytes                                       RW
```

Synopsis

Used to assign the Power Data Object (PDO) current (mA) to the selected source PDO when the power supply type is fixed or variable. This is can be called only after a source PDO has been selected and its type has been assigned.

5.17.30 TSI_USBC_PWR_LOCAL_SOURCE_PDO_VOLTAGE

```
TSI_USBC_PWR_LOCAL_SOURCE_PDO_VOLTAGE          0xfe4
unsigned int Source_PDO_voltage                U32
4 bytes                                        RW
```

Synopsis

Used to assign the Power Data Object (PDO) voltage (mV) to the selected source PDO when the power supply type is fixed. This is can be called only after a source PDO has been selected and its type has been assigned.

5.17.31 TSI_USBC_PWR_LOCAL_SOURCE_PDO_PEAK_CURRENT

```
TSI_USBC_PWR_LOCAL_SOURCE_PDO_PEAK_CURRENT     0xfe5
unsigned int Source_PDO_peak_current           U32
4 bytes                                        RW
```

Synopsis

Used to assign the Power Data Object (PDO) overload capabilities (list here as 100%, 110%, 125% and 150%) to the selected PDO when the power supply type is fixed. This is can be called only after a source PDO has been selected and its type has been assigned. These values actually represent bits 20 and 21in PDO specifications.

5.17.32 TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_POWER

```
TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_POWER        0xfe6
unsigned int Source_PDO_max_power              U32
4 bytes                                        RW
```

Synopsis

Used to assign the Power Data Object (PDO) maximum power (mW) to the selected source PDO when the power supply type is battery. This is can be called only after a source PDO has been selected and its type has been assigned.

5.17.33 TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_VOLTAGE

```
TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_VOLTAGE     0xfe7
unsigned int Source_PDO_max_voltage           U32
4 bytes                                       RW
```

Synopsis

Used to assign the Power Data Object (PDO) maximum voltage (mV) to the selected source PDO when the power supply type is variable or battery. This is can be called only after a source PDO has been selected and its type has been assigned.

5.17.34 TSI_USBC_PWR_LOCAL_SOURCE_PDO_MIN_VOLTAGE

```
TSI_USBC_PWR_LOCAL_SOURCE_PDO_MIN_VOLTAGE     0xfe8
unsigned int Source_PDO_min_voltage           U32
4 bytes                                       RW
```

Synopsis

Used to assign the Power Data Object (PDO) minimum voltage (mV) to the selected source PDO when the power supply type is variable or battery. This is can be called only after a source PDO has been selected and its type has been assigned.

5.17.35 TSI_USBC_PWR_LOCAL_SOURCE_PDO_FIXED_SUPPLY_BITS_25_TO_29

```
TSI_USBC_PWR_LOCAL_SOURCE_PDO_FIXED_SUPPLY_BITS_25_TO_29
                                              0xff0
unsigned int Source_PDO_fixed_bits            U32
4 bytes                                       RW
```

Synopsis

Used to assign the Power Data Object (PDO) bits 25 through 29 to the selected source PDO when the power supply type is fixed. This is can be called only after a source PDO has been selected and its type has been assigned.

Bit	Description
25	Dual-Role Data
26	USB Communications Capable
27	Unconstrained Power
28	USB Suspend Supported

29	Dual-Role Power

5.17.36 TSI_USBC_PWR_LOCAL_SINK_PDO_SELECT

```
TSI_USBC_PWR_LOCAL_SINK_PDO_SELECT          0xfe9
unsigned int Sink_PDO_select                U32
4 bytes                                     RW
```

Synopsis

Used to select the sink Power Data Object (PDO) by number (0-6) to be added or edited in USB-C enabled devices. This is the first step when we wish to add or modify a sink PDO.

5.17.37 TSI_USBC_PWR_LOCAL_SINK_PDO_TYPE

```
TSI_USBC_PWR_LOCAL_SINK_PDO_TYPE            0xfea
unsigned int Sink_PDO_type                  U32
4 bytes                                     RW
```

Synopsis

Used to assign the Power Data Object (PDO) type (0-3) to the selected sink PDO (via the TSI_USBC_PWR_LOCAL_SOURCE_PDO_SELECT CI above). Values are: 0 to disable, 1 for fixed power supply, 2 for variable power suppy and 3 for battery. This is the second step when we wish to add or modify a PDO. An error will occur if no PDO sink object is selected.

5.17.38 TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_CURRENT

```
TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_CURRENT     0xfeb
unsigned int Sink_PDO_max_current           U32
4 bytes                                     RW
```

Synopsis

Used to assign the Power Data Object (PDO) current (mA) to the selected sink PDO when the power supply type is fixed or variable. This is can be called only after a sink PDO has been selected and its type has been assigned.

5.17.39 TSI_USBC_PWR_LOCAL_SINK_PDO_VOLTAGE

```
TSI_USBC_PWR_LOCAL_SINK_PDO_VOLTAGE          0xfec
unsigned int Sink_PDO_voltage                U32
4 bytes                                      RW
```

Synopsis

Used to assign the Power Data Object (PDO) voltage (mV) to the selected sink PDO when the power supply type is fixed. This is can be called only after a sink PDO has been selected and its type has been assigned.

5.17.40 TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_POWER

```
TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_POWER        0xfed
unsigned int Sink_PDO_max_power              U32
4 bytes                                      RW
```

Synopsis

Used to assign the Power Data Object (PDO) maximum power (mW) to the selected sink PDO when the power supply type is battery. This is can be called only after a sink PDO has been selected and its type has been assigned.

5.17.41 TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_VOLTAGE

```
TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_VOLTAGE      0xfee
unsigned int Sink_PDO_max_voltage            U32
4 bytes                                      RW
```

Synopsis

Used to assign the Power Data Object (PDO) maximum voltage (mV) to the selected sink PDO when the power supply type is variable or battery. This is can be called only after a sink PDO has been selected and its type has been assigned.

5.17.42 TSI_USBC_PWR_LOCAL_SINK_PDO_MIN_VOLTAGE

```
TSI_USBC_PWR_LOCAL_SINK_PDO_MIN_VOLTAGE      0xfef
unsigned int Sink_PDO_min_voltage            U32
4 bytes                                      RW
```

Synopsis

Used to assign the Power Data Object (PDO) minimum voltage (mV) to the selected sink PDO when the power supply type is variable or battery. This is can be called only after a sink PDO has been selected and its type has been assigned.

5.17.43 TSI_USBC_PWR_LOCAL_SINK_PDO_FIXED_SUPPLY_BITS_25_TO_29

```
TSI_USBC_PWR_LOCAL_SINK_PDO_FIXED_SUPPLY_BITS_25_TO_29
                                                0xff1
unsigned int Sink_PDO_fixed_bits                U32
4 bytes                                         RW
```

Synopsis

Used to assign the Power Data Object (PDO) bits 25 through 29 to the selected sink PDO when the power supply type is fixed. This is can be called only after a sink PDO has been selected and its type has been assigned. Note that bit 28 Higher Capability should be set when more than one sink PDO is made available.

Bit	Description
25	Dual-Role Data
26	USB Communications Capable
27	Unconstrained Power
28	Higher Capability
29	Dual-Role Power

5.17.44 TSI_USBC_RESISTANCE_CTRL

```
TSI_USBC_RESISTANCE_CTRL                        0x616
unsigned int resistance_ctrl                    U32
4 bytes                                         RW
```

Synopsis

Set resistance to be turned on. Returns error in case if specified resistance is not available. Set value to 0 for disabling the resistance.

Get closest higher resistance available if previous set call was unsuccessful. Otherwise returns 0.

Bits	Description	
0	Type of call:	
	0	Test appliance of specified resistance. Resistance will not be applied. An error will be returned in case if resistance is not applicable and caller shall read this CI once again to obtain closest higher resistance available for use.
	1	Resistance value to be immediately applied, if correct.
1	Type of resistor:	
	0	Internal.
	1	External.
31:2	Resistance value in mΩ (milliohms). Maximum possible value is 1073741823 mΩ (1.073741823 MΩ).	

5.17.45 TSI_R_USBC_INT_RESISTANCE_STATUS

```
TSI_USBC_INT_RESISTANCE_STATUS                0x617
unsigned int resistance_status                U32
4 bytes                                       RO
```

Synopsis

Get currently enabled internal resistance in mΩ (milliohms). Returns 0 if no resistance set.

5.17.46 TSI_R_USBC_EXT_RESISTANCE_STATUS

```
TSI_USBC_EXT_RESISTANCE_STATUS                0x617
unsigned int resistance_status                U32
4 bytes                                       RO
```

Synopsis

Get currently enabled external resistance in mΩ (milliohms). Returns 0 if no resistance set.

5.18 Pattern generator CI definitions

ClientVersion 11, and higher *<license:TBD>*

This section defines the new configuration items that are specific to pattern generator functionality in TSI. The CI Space from 0x700 to 0x77f is reserved for pattern generator specific controls.

5.18.1 TSI_PG_ENABLED_STREAM_COUNT

```
TSI_PG_ENABLED_STREAM_COUNT                    0x700
unsigned int enabled_stream_count              U32
4 bytes                                        RW
```

Synopsis

Used to get or set number of pattern generators running. Maximum number of supported streams depends on used video interface, and is available as run-time information by reading the 5.18.2 TSI_R_PG_MAX_STREAM_COUNT CI. Zero means no pattern generators are running. By default, output interfaces have one stream enabled.

Important: *This setting does have effect between SST/MST selection. To enable (or disable) MST, please refer to 5.15.7 TSI_W_DPTX_COMMAND.*

5.18.2 TSI_R_PG_MAX_STREAM_COUNT

```
TSI_R_PG_MAX_STREAM_COUNT                      0x701
unsigned int max_stream_count                  U32
4 bytes                                        RO
```

Synopsis

Indicates the maximum number of streams the currently selected interface can support. Zero means no pattern generators can be used on the interface.

5.18.3 TSI_PG_STREAM_SELECT

```
TSI_PG_STREAM_SELECT                          0x702
unsigned int stream_select                    U32
4 bytes                                       RW
```

Synopsis

Get or set the stream to be configured. Streams are numbered from zero to max number of streams (5.18.2 TSI_R_PG_MAX_STREAM_COUNT) minus one. For SST only interfaces reading or writing this selection has no effect.

Important: *You can select streams that are not enabled (yet) in order to configure the parameters before enabling the stream.*

5.18.4 TSI_W_PG_COMMAND

```
TSI_W_PG_COMMAND                              0x703
unsigned int pattern_gen_command              U32
4 bytes                                       WO
```

Synopsis

Write non-zero value to apply current timing and pattern on the active output. Settings will be applied on all active streams. Number of enabled streams is configured with CI 5.18.1 TSI_PG_ENABLED_STREAM_COUNT.

5.18.5 TSI_PG_CUSTOM_TIMING_HTOTAL

```
TSI_PG_CUSTOM_TIMING_HTOTAL                   0x704
unsigned int custom_timing_htotal             U32
4 bytes                                       RW
```

Synopsis

Indicates the total width of a scanline in pixel clocks.

5.18.6 TSI_PG_CUSTOM_TIMING_HSTART

```
TSI_PG_CUSTOM_TIMING_HSTART                    0x705
unsigned int custom_timing_hstart              U32
4 bytes                                        RW
```

Synopsis

Defines number of pixels from end of H-Sync to start of active area.

5.18.7 TSI_PG_CUSTOM_TIMING_HACTIVE

```
TSI_PG_CUSTOM_TIMING_HACTIVE                   0x706
unsigned int custom_timing_hacktive            U32
4 bytes                                        RW
```

Synopsis

Indicates the width of video frame that is visible on screen as number of pixel clocks.

5.18.8 TSI_PG_CUSTOM_TIMING_HSYNCW

```
TSI_PG_CUSTOM_TIMING_HSYNCW                    0x707
int custom_timing_hsyncw                       I32
4 bytes                                        RW
```

Synopsis

Indicates the width of horizontal sync as number of pixel clocks.

Important: *Writing negative value means negative polarity of this sync signal.*

5.18.9 TSI_PG_CUSTOM_TIMING_VTOTAL

```
TSI_PG_CUSTOM_TIMING_VTOTAL                    0x708
unsigned int custom_timing_vtotal              U32
4 bytes                                        RW
```

Synopsis

Indicates the total height of the frame as number of scanlines.

5.18.10 TSI_PG_CUSTOM_TIMING_VSTART

```
TSI_PG_CUSTOM_TIMING_VSTART                    0x709
unsigned int custom_timing_vstart              U32
4 bytes                                        RW
```

Synopsis

Defines number of scan lines from end of V-Sync to start of active area.

5.18.11 TSI_PG_CUSTOM_TIMING_VACTIVE

```
TSI_PG_CUSTOM_TIMING_VACTIVE                   0x70a
unsigned int custom_timing_vactive             U32
4 bytes                                        RW
```

Synopsis

Indicates the height of the frame that is visible on screen as number of scanlines.

5.18.12 TSI_PG_CUSTOM_TIMING_VSYNCW

```
TSI_PG_CUSTOM_TIMING_VSYNCW                    0x70b
int custom_timing_VSYNCW                       I32
4 bytes                                        RW
```

Synopsis

Indicates the width of the vertical sync as number of scanlines.

Important: *Writing negative value means negative polarity of this sync signal.*

5.18.13 TSI_PG_CUSTOM_TIMING_FLAGS

```
TSI_PG_CUSTOM_TIMING_FLAGS                    0x70c
unsigned int custom_timing_flags              U32
4 bytes                                       RW
```

Synopsis

Signal output timing flags and color information. These settings are used to set-up the physical output signal. Please see table below:

Bits	Description	
7:0	Color depth	
	0	6 bits per color channel
	1	8 bits per color channel
	2	10 bits per color channel
	3	12 bits per color channel
	4	16 bits per color channel
	*	RESERVED
8	Interlace	
	0	Progressive
	1	Interlaced
9	H-Sync polarity ***Important:*** *TSI LITE edition does not support this flag. Please use negative H-Sync value instead.*	
	0	Positive
	1	Negative
10	V-Sync polarity ***Important:*** *TSI LITE edition does not support this flag. Please use negative V-sync value instead.*	
	0	Positive
	1	Negative
14:11	Output color space	
	0	RGB
	1	YCbCr 4:4:4.
	2	YCbCr 4:2:2.
	3	YCbCr 4:2:0.
	*	RESERVED
15	RESERVED	
19:16	Output colorimetry	
	0	ITU-701
	1	ITU-609
31:20	RESERVED	

5.18.14 TSI_PG_CUSTOM_TIMING_FIELD_RATE

```
TSI_PG_CUSTOM_TIMING_FIELD_RATE                0x70d
unsigned int custom_timing_field_rate          U32
4 bytes                                        RW
```

Synopsis

Indicates the field rate as mHz (1000 = 1Hz). For non-interlaced timings, the field rate is same as frame rate.

5.18.15 TSI_R_PG_PREDEF_TIMING_COUNT

```
TSI_R_PG_PREDEF_TIMING_COUNT                   0x70e
unsigned int predef_timing_count               U32
4 bytes                                        RO
```

Synopsis

Indicates the number of predefined timings available for the current interface. Please notice that the number of timings depends on the type of device being used, and on the interface being used.

5.18.16 TSI_W_PG_PREDEF_TIMING_SELECT

```
TSI_PG_PREDEF_TIMING_SELECT                    0x70f
unsigned int predef_timing_select              U32
4 bytes                                        WO
```

Synopsis

Selects a predefined timing. Writing into this CI will update the custom timing values as defined by the timing. The custom timing information is therefore made readable to the client application.

Important: *Selecting a custom timing does not apply it to the patter generator. To apply a predefined timing, first select the timing using this CI, and then issue apply new timing command with the 5.18.4 TSI_W_PG_COMMAND CI.*

5.18.17 TSI_PG_PREDEF_PATTERN_COUNT

```
TSI_PG_PREDEF_PATTERN_COUNT                    0x710
unsigned int predef_pattern_count              U32
4 bytes                                        RO
```

Synopsis

Indicates number of predefined patterns available for the current interface. Please notice that the number of patterns depends on the type of device being used, and of the interface being used.

5.18.18 TSI_W_PG_PREDEF_PATTERN_SELECT

```
TSI_PG_PREDEF_PATTERN_SELECT                   0x711
unsigned int predef_pattern_select             U32
4 bytes                                        WO
```

Synopsis

Select a predefined pattern for use and/or enumeration.

Important: *Selecting a pattern does not apply it on the pattern generator. To apply a predefined pattern, first select the pattern using this CI, and then issue apply selected predefined pattern command with the 5.18.4 TSI_W_PG_COMMAND CI.*

Important: *Selecting a pattern clears the currently assigned custom pattern and data. If custom pattern data is present, it is always used instead of any predefine pattern.*

5.18.19 TSI_R_PG_PREDEF_PATTERN_NAME

```
TSI_R_PG_PREDEF_PATTERN_NAME                   0x712
char predef_pattern_name[]                     ARRAY_U8
Variable size, max size 256                    RO
```

Synopsis

Contains an ASCII formatted string indicating the name of the currently selected predefined pattern.

5.18.20 TSI_R_PG_PREDEF_PATTERN_ID

```
TSI_R_PG_PREDEF_PATTERN_ID                    0x713
unsigned int predef_pattern_id                U32
4 bytes                                       RO
```

Synopsis

Contains pattern identifier as a computer friendly way. The pattern ID's are used to identify which predefined pattern parameters can be used with which pattern(s).

Important: *The pattern ID's are unique per parameters, meaning that not all patterns have a unique pattern ID.*

5.18.21 TSI_PG_PREDEF_PATTERN_PARAMS

```
TSI_PG_PREDEF_PATTERN_PARAMS                  0x714
unsigned int predef_pattern_params            ARRAY_U32
Variable size                                 RW
```

Synopsis

Used to access pattern specific parameters that can be used to alter the appearance of procedurally generated patterns. Please read the 5.18.20 TSI_R_PG_PREDEF_PATTERN_ID CI to determine what type of parameters the pattern accepts. See table below for currently defined parameter sets:

ID	Size (Bytes)	Word Index	Description
0	0		No parameters
1	8	0	Color row width in pixels. Default value is 1
		1	Black row width in pixels. Default value is 1
2	4	0	Color step. Default value is 100
3	4	0	Number of frames. Default value is 1

Important: *Changing the pattern parameters does not change the visible pattern. To change parameters, select the predefined pattern first, then update the parameters and after that issue "apply selected predefined pattern" command on the 5.18.4 TSI_W_PG_COMMAND CI.*

5.18.22 TSI_PG_CUSTOM_PATTERN_WIDTH

```
TSI_PG_CUSTOM_PATTERN_WIDTH                    0x715
unsigned int custom_pat_width                  U32
4 bytes                                        RW
```

Synopsis

Indicates width of custom pattern bitmap as number of pixels.

5.18.23 TSI_PG_CUSTOM_PATTERN_HEIGHT

```
TSI_PG_CUSTOM_PATTERN_HEIGHT                   0x716
unsigned int custom_pat_height                 U32
4 bytes                                        RW
```

Synopsis

Indicates height of custom pattern bitmap as number of pixels.

5.18.24 TSI_PG_CUSTOM_PATTERN_PIXEL_FORMAT

```
TSI_PG_CUSTOM_PATTERN_PIXEL_FORMAT          0x717
unsigned int custom_pat_pxl_format          U32
4 bytes                                     RW
```

Synopsis

Defines which data format is delivered to TSI through TSI_PG_CUSTOM_PATTERN_DATA.

Important: *This definition is independent from TSI_PG_CUSTOM_TIMING_FLAGS, which defines the actual output video stream format. This setting defines the bitmap data format as it is arranged in computer memory.*

Important: *The correct way of filling up the pixel data is to "left-align" the effective bits. i.e. the most significant bit in pixel data will be most significant bit also in the output video stream regardless of how many bits there are in the video stream output compared to the pixel data.*

Important: *TSI can perform conversions between bit-depths and formats of same color space; However, the provided custom bitmap data must match the color space of the physical output. If these settings do not match, the output is blanked.*

Important: *The YCbCr "4:4:4", "4:2:2" and "4:2:0" data formats have different definitions, but are considered same color-space, and therefore can be converted into each others by TSI.*

Value	Description
0x000	RGB 8:8:8, 24-bit RGB image. Stored as 3 bytes. Lowest memory location stores "R" channel, and highest "B" channel
0x001	RGB 16:16:16, 48-bit RGB image. Stored as 3 little-endian 16-bit words. Lowest memory location stores "R" channel, and highest "B" channel.
0x100	YCbCr 8:8:8, 24-bit YCbCr "4:4:4" image. Stored as 3 bytes. Lowest memory location stores "Y" channel, and highest "Cr" channel
0x101	YCbCr 16:16:16, 48-bit YCbCr "4:4:4" image. Stored as 3 little-endian 16-bit words. Lowest memory location stores "Y" channel, and highest "Cr" channel.
0x200	YCbYCr 8:8:8:8, "16-bit" YCbCr "4:2:2" image. Stored as 4 bytes. Lowest memory location stores first "Y" value, and highest "Cr" channel. Each set of four bytes has data for two pixels.
0x201	YCbYCr 16:16:16:16, "32-bit" YCbCr "4:2:2" image. Stored as 4 little-endian 16-bit words. Lowest memory location stores first "Y" value, and highest "Cr" channel. Each set of four words has data for two pixels.
0x300	YYCbYYCr 8:8:8:8:8:8, "12-bit" YCbCr "4:2:0" image. Stored as 6 bytes. Lowest memory location stores the first "Y" value, and highest memory location stores "Cr" channel. Each set of six bytes has data for four pixels arranged as 2x2 block in output image. Y-values are filled in-order from left to right, and from top to bottom.
0x301	YYCbYYCr 16:16:16:16:16:16, "24-bit" YCbCr "4:2:0" image. Stored as 6 little-endian 16-bit words. Lowest memory location stores the first "Y" value, and highest memory location stores "Cr" channel. Each set of six words has data for four pixels arranged as 2x2 block in output image. Y-values are filled in-order from left to right, and from top to bottom.
0x8000 to 0xffff	ID values reserved for device specific formats. Device specific formats are NEVER converted into any other format, and require a matching setting in TSI_PG_CUSTOM_TIMING_FLAGS to be visible on the output.
*	RESERVED

5.18.25 TSI_PG_CUSTOM_PATTERN_DATA

```
TSI_PG_CUSTOM_PATTERN_DATA                    0x718
unsigned char pattern_data                    ARRAY_U8
Variable size                                 RW
```

Synopsis

Contains RAW data that makes up the custom pattern as defined in CI's TSI_PG_CUSTOM_PATTERN_WIDTH, TSI_PG_CUSTOM_PATTERN_HEIGHT and TSI_PG_CUSTOM_PATTERN_PIXEL_FORMAT. If any valid custom pattern data is assigned, it will be used when the timing/pattern is applied next time by issuing update command by writing into TSI_W_PG_COMMAND. Is used for uploading one custom pattern bitmap into one selected memory block.

Important: *Writing into TSI_W_PG_PREDEF_TIMING_SELECT will clear custom pattern definitions and data. The custom pattern can also be cleared by writing zero bytes of data into TSI_PG_CUSTOM_PATTERN_DATA.*

5.18.26 TSI_PG_CUSTOM_PATTERN_INDEX

```
TSI_PG_CUSTOM_PATTERN_INDEX                   0x719
unsigned int custom_pat_index                 U64
8 bytes                                       RW
```

Synopsis

Indicates memory block that contains custom pattern bitmap data.

5.18.27 TSI_PG_CUSTOM_PATTERN_CHAINED_DATA

```
TSI_PG_CUSTOM_PATTERN_CHAINED_DATA            0x720
unsigned char pattern_data                    ARRAY_U8
Variable size                                 RW
```

Synopsis

Contains RAW data that makes up the custom pattern as defined in CI's TSI_PG_CUSTOM_PATTERN_WIDTH, TSI_PG_CUSTOM_PATTERN_HEIGHT and TSI_PG_CUSTOM_PATTERN_PIXEL_FORMAT. If any valid custom pattern data is assigned, it will be used when the timing/pattern is applied next time by issuing update command by writing into TSI_W_PG_COMMAND. Is used for uploading multiple custom pattern bitmaps of the same size into one selected memory block.

Important: *Writing into TSI_W_PG_PREDEF_TIMING_SELECT will clear custom pattern definitions and data. The custom pattern can also be cleared by writing zero bytes of data into TSI_PG_CUSTOM_PATTERN_DATA.*

5.18.28 TSI_PG_CUSTOM_PATTERN_FRAME_TYPE

```
TSI_PG_CUSTOM_PATTERN_FRAME_TYPE                0x721
unsigned int custom_pat_frame                   U32
4 bytes                                         RW
```

Synopsis

Indicates frame size used for displaying custom pattern bitmap.

Value	Description
0x000	Uses actual frame size according to previously set custom pattern bitmap resolution
0x002	Uses maximum possible frame size according to the device type (4K for UCD-3xx family, 8K for UCD-400)
*	RESERVED

5.19 Scripting support

This section lists CI's that are intended to be used with TSI Scripts.

Define	Config ID	Default	Description	Reference
TSI_W_SCRI_DELAY	0x70000001	N/A	Generates delay on write. The delay is the written value in milliseconds.	

5.19.1 TSI_W_SCRI_DELAY

```
TSI_W_SCRI_DELAY                                0x70000001
unsigned int script_delay_on_w                  U32
4 bytes                                         WO
```

Synopsis

This CI is intended for scripts (such as the ones used with TSI_TST_Init and TSI_TST_Run) that were added to support TestStand [TestStand is a trademark of National Instruments Corp.]. The CI will delay the TSI_TS_SetConfigItem(…) function call. The delay is in milliseconds and it's length equal to the value being written.

5.20 Video timing detail CI's

ClientVersion 7, and higher *No license requirements*

This section contains definitions for configuration items used for reading input video timing details from DP MSA and/or HDMI Video information.

Name	Description	Reference
TSI_W_DPRX_MSA_COMMAND	Used to update MSA DATA from the device.	5.20.1
TSI_R_DPRX_MSA_STREAM_COUNT	Number of streams from previous update	5.20.2
TSI_R_DPRX_MSA_DATA	Allows access to MSA data from the device as it is received from the device.	5.20.3
TSI_DPRX_MSA_STREAM_SELECT	Select which MST stream to read the MSA data from.	5.20.4
TSI_R_DPRX_MSA_N_VIDEO	Used in DP link clock calculation	5.20.5
TSI_R_DPRX_MSA_M_VIDEO	Used in DP link clock calculation	5.20.6
TSI_R_DPRX_MSA_HTOTAL	Scanline total pixel count.	5.20.7
TSI_R_DPRX_MSA_VTOTAL	Frame total scanline count.	5.20.8
TSI_R_DPRX_MSA_HACTIVE	Horizontal active area size as count of pixels.	5.20.9
TSI_R_DPRX_MSA_VACTIVE	Vertical active area size as count of scanlines.	5.20.10
TSI_R_DPRX_MSA_HSYNC_WIDTH	Horizontal sync width as count of pixels.	5.20.11
TSI_R_DPRX_MSA_VSYBC_WIDTH	Vertical sync width as count of scanlines.	5.20.12
TSI_R_DPRX_MSA_HSTART	Horizontal start position of active area as count of pixels.	5.20.13
TSI_R_DPRX_MSA_VSTART	Vertical start position of active area as count of scanlines.	5.20.14
TSI_R_DPRX_MSA_MISC	MSA Misc bits.	5.20.15
TSI_R_DPRX_MSA_VBID	VBID	5.20.16
TSI_R_DPRX_MSA_PORT_NUMBER	Port number	5.20.17

5.20.1 TSI_W_DPRX_MSA_COMMAND

```
TSI_W_DPRX_MSA_COMMAND                          0x260
unsigned int msa_command                        U32
4 bytes                                         WO
```

Description

Write "1" to read new MSA data from the device. The update command update all the read only MSA fields, and reset the MSA stream selection to zero.

5.20.2 TSI_R_DPRX_MSA_STREAM_COUNT

```
TSI_R_DPRX_MSA_STREAM_COUNT                    0x261
unsigned int msa_stream_count                  U32
4 bytes                                        RO
```

Description

Number of MSA parameter blocks received from the device on previous update command.

5.20.3 TSI_R_DPRX_MSA_DATA

```
TSI_R_DPRX_MSA_DATA                            0x262
unsigned int msa_data[]                        ARRAY_U32
Variable size                                  RO
```

Description

Allows reading native MSA data as it was received from the device. All available MSA data blocks can be read with a single read. Reading this CI will always return data starting from the first MSA data block regardless of value in TSI_DRPX_MSA_STREAM_SELECT. Each MSA data block consists of 7 unsigned integers (32-bit values). Please see table below describing the contents of each data MSA data block:

Array Index	Bits	Description
0	23:0	N Video.
1	23:0	M Video.
2	15:0	H-Total.
	31:16	V-Total.
3	15:0	H-Active.
	31:16	V-Active.
4	15:0	H-Sync width – Negative value indicates negative H-Sync polarity.
	31:16	V-Sync width – Negative value indicates negative V-Sync polarity.
5	15:0	H-Sync start.
	31:16	V-Sync start.
6	7:0	MSA MISC-0 bits.
	15:8	MSA MISC-1 bits.
	23:16	VBID
	30:24	Port number

Important: *Undefined bits are reserved, and should be ignored.*

5.20.4 TSI_DPRX_MSA_STREAM_SELECT

```
TSI_DPRX_MSA_STREAM_SELECT                    0x263
unsigned int msa_stream_select                U32
4 bytes                                       RW
```

Description

Used to select which MSA data block is accessed with field decoded CI's. MSA Blocks are numbered starting from zero (0), up to number of available streams minus one. Please read TSI_R_DPRX_MSA_STREAM_COUNT to determine number of streams.

5.20.5 TSI_R_DPRX_MSA_N_VIDEO

```
TSI_R_DPRX_MSA_N_VIDEO                        0x264
unsigned int msa_n_video                      U32
4 bytes                                       RO
```

Description

Reads MSA_N_VIDEO field of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

5.20.6 TSI_R_DPRX_MSA_M_VIDEO

```
TSI_R_DPRX_MSA_M_VIDEO                        0x265
unsigned int msa_m_video                      U32
4 bytes                                       RO
```

Description

Reads MSA_M_VIDEO field of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

5.20.7 TSI_R_DPRX_MSA_HTOTAL

```
TSI_R_DPRX_MSA_HTOTAL                         0x266
unsigned int msa_htotal                       U32
4 bytes                                       RO
```

Description

Reads MSA_HTOTAL field of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

5.20.8 TSI_R_DPRX_MSA_VTOTAL

```
TSI_R_DPRX_MSA_VTOTAL                          0x267
unsigned int msa_vtotal                        U32
4 bytes                                        RO
```

Description

Reads MSA_VTOTAL field of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

5.20.9 TSI_R_DPRX_MSA_HACTIVE

```
TSI_R_DPRX_MSA_HACTIVE                         0x268
unsigned int msa_hactive                       U32
4 bytes                                        RO
```

Description

Reads MSA_HACTIVE field of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

5.20.10 TSI_R_DPRX_MSA_VACTIVE

```
TSI_R_DPRX_MSA_VACTIVE                         0x269
unsigned int msa_vactive                       U32
4 bytes                                        RO
```

Description

Reads MSA_VACTIVE field of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

5.20.11 TSI_R_DPRX_MSA_HSYNC_WIDTH

```
TSI_R_DPRX_MSA_HSYNC_WIDTH                     0x26a
int msa_hsync_width                            S32
4 bytes                                        RO
```

Description

Reads MSA_HSYNC_WIDTH field of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

Important: *Negative HSYNC_WIDTH indicates negative H-Sync polarity.*

5.20.12 TSI_R_DPRX_MSA_VSYNC_WIDTH

```
TSI_R_DPRX_MSA_VSYNC_WIDTH                    0x26b
int msa_vsync_width                           S32
4 bytes                                       RO
```

Description

Reads MSA_VSYNC_WIDTH field of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

Important: *Negative VSYNC_WIDTH indicates negative V-Sync polarity.*

5.20.13 TSI_R_DPRX_MSA_HSTART

```
TSI_R_DPRX_MSA_HSTART                         0x26c
unsinged int msa_hstart                       U32
4 bytes                                       RO
```

Description

Reads MSA_HSTART field of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

5.20.14 TSI_R_DPRX_MSA_VSTART

```
TSI_R_DPRX_MSA_VSTART                         0x26d
unsigned int msa_vstart                       U32
4 bytes                                       RO
```

Description

Reads MSA_VSTART field of the MSA data block selected with TSI_DPRX_STREAM_SELECT CI.

5.20.15 TSI_R_DPRX_MSA_MISC

```
TSI_R_DPRX_MSA_MISC                           0x26e
unsigned int msa_misc                         U32
4 bytes                                       RO
```

Description

Reads MSA_MISC field (MISC0 in bits 7:0 and MISC1 in bits 15:8) of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

5.20.16 TSI_R_DPRX_MSA_VBID

```
TSI_R_DPRX_MSA_VBID                                 0x26f
unsigned int msa_vbid                               U32
4 bytes                                             RO
```

Description

Reads MSA_VBID field of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

5.20.17 TSI_R_DPRX_MSA_PORT_NUMBER

```
TSI_R_DPRX_MSA_PORT_NUMBER                          0x270
unsigned int msa_port_number                        U32
4 bytes                                             RO
```

Description

Reads MSA_PORT_NUMBER field of the MSA data block selected with TSI_DPRX_MSA_STREAM_SELECT CI.

6 TESTS

All tests, and their associated configurations and requirements are listed here.

6.1 Compare video frame sequence with a single reference

ClientVersion 1, and later *Basic license required*

```
#define TSI_TEST_VIDEO_PXL_TOLERANCE    2     // TEST ID
```

Synopsis

Compare a defined number of captured frames to a single reference frame. The test will capture the required number of consecutive frames into system RAM and then perform analysis between each frame and reference frame. Test is considered passed, if the number of failed frames does not exceed the programmed value.

Configuration items

The following table defines the configuration items which are always used during the video test:

Define	Config ID	Default	Description	Reference
TSI_REF1_WIDTH	0x10	-	Reference width as count of elements	6.1.1
TSI_REF1_HEIGHT	0x11	-	Reference height as count of elements	6.1.2
TSI_REF1_ELEMENT_SIZE	0x12	-	Size of a single element as bytes	6.1.3
TSI_REF1_ELEMENT_WIDTH	0x13	-	Element width as pixels	6.1.4
TSI_REF1_ELEMENT_HEIGHT	0x14	-	Element height as pixels	6.1.5
TSI_REF1_COLOR_DEPTH	0x15	-	Active color bits per channel	6.1.6
TSI_REF1_ELEMENT_FORMAT	0x16	-	Element data layout ID	6.1.7
TSI_REF1_FRAME_DATA	0x17	-	Array of bytes that form the bitmap	6.1.8
TSI_TEST_LENGTH	0x1000	60	Length of video test in frames	6.1.9
TSI_LIM_FRAME_MISMATCHES	0x1001	0	Maximum number of failed frames allowed per test	6.1.10
TSI_LIM_PIXEL_MISMATCHES	0x1002	0	Maximum number of failed pixels allowed per frame	6.1.11
TSI_PIXEL_TOLERANCE	0x1003	0	Maximum difference between color-values allowed.	6.1.12

(Continued...)

(...Continued)

The following table defines configuration items that are used to control optional features of the video test:

Define	Config ID	Default	Description	Reference
TSI_MAX_AUTO_SAVE_FAILED	0x1080	0	Maximum number of failed frames to auto-save.	6.1.13
TSI_FAILED_FRAME_TARGET_FOLDER	0x1081	-	Location where the failed frames are to be saved	6.1.14
TSI_MAX_EXPORT_FAILED	0x1082	0	Maximum number of exported frames	6.1.15
TSI_R_VIDEO_TEST_RAW_RESULTS_DATA	0x1008	-	Comparison error counters	6.1.16
TSI_EXPORTED	0x108e	-	Number of frames exported	6.1.17
TSI_EXPORT_ACCESS_INDEX	0x108f	-	Defines which exported frame to access	6.1.18
TSI_EXPORT_WIDTH	0x1090	-	Width of exported frame as count of elements	6.1.19
TSI_EXPORT_HEIGHT	0x1091	-	Height of exported frame as count of elements	6.1.20
TSI_EXPORT_ELEMENT_SIZE	0x1092	-	Size of an element as count of bytes	6.1.21
TSI_EXPORT_ELEMENT_WIDTH	0x1093	-	Width of an element as pixels	6.1.22
TSI_EXPORT_ELEMENT_HEIGHT	0x1094	-	Height of an element as pixels	6.1.23
TSI_EXPORT_COLOR_DEPTH	0x1095	-	Color depth per color as count of bits	6.1.24
TSI_EXPORT_ELEMENT_FORMAT	0x1096	-	Element data layout ID	6.1.25
TSI_EXPORT_FRAME_DATA	0x1097	-	Exported frame bitmap data.	6.1.26

Log messages

The test run is divided into three (3) stages.

Stage one is initialization, resource allocation and basic check of test parameters. The most important test parameters are logged. If no problems are detected, the test proceeds to stage two. Example log:

```
Starting video test (Test ID 2)
Stage 1: Test initialization. Test params:
  - Test length 60
  - Reference Width = 640
  - Reference Height = 480
  - Reference Element Width = 1
  - Reference Element Height = 1
  - Reference Format = 0
  - Reference Element byte size = 3
  - Reference Bit per channel = 8
```

Stage two is data gathering. During this stage the frames to be tested are simply captured into system RAM for the next stage. Example log:

```
Stage 1 Completed -- Entering stage 2: Data gathering
Stage 2 Completed -- Entering Stage 3: Compare and analysis
```

(Continued...)

(...Continued)

Stage three is analysis. Each frame is compared to the reference frame, and the frame analysis results are logged:

```
Stage 3, Frame 1 analysis results:
  - Failed pixels per sub channel: Red = 201914, Green = 201914, Blue = 201808
  - Total pixel errors = 269710, Highest deviation = 255
  - Mean deviation of pixels = 10.106
  - Total failed pixel errors exceed allowed pixel errors (0): BAD frame
  - The number of total frames exceed allowed bad frames (0).
Stage 3 completed -- Test failed
```

What the above actually means is this: There are 201914 pixels with errors on the red color channel, 201914 pixels with errors on green color channel and 201808 pixels with errors on the blue color channel.

269710 Pixels had error within the pixel (on at least one of three color channels). This value is at least equal to highest color-channel specific error count, and it can be as high as all color channel specific error counts combined. The Highest deviation tells the highest difference on any color channel between reference frame and compared frame.

Mean deviation is calculated by adding all deviations to together and dividing by the number of pixels in the frame.

The number of failed pixels exceed the number of allowed failures (which was zero), so the frame is considered “bad”.

The number of bad frames exceeds the number of allowed bad frames (which was also zero).

Thus the test outcome is “failed”.

6.1.1 TSI_REF1_WIDTH

```
TSI_REF1_WIDTH                                  0x10
unsigned int Ref1W                              U32
4 bytes                                         RW
```

Description

Defines frame width as number of elements. Actual width in pixels is therefore this value multiplied by the element width.

6.1.2 TSI_REF1_HEIGHT

```
TSI_REF1_HEIGHT                                 0x11
unsigned int Ref1H                              U32
4 bytes                                         RW
```

Description

Defines frame height as number of elements. Actual height in pixels is therefore this value multiplied by the element height.

6.1.3 TSI_REF1_ELEMENT_SIZE

```
TSI_REF1_ELEMENT_SIZE                          0x12
unsigned int Ref1ElementSize                   U32
4 bytes                                        RW
```

Description

Defines the size of the element body, in as bytes of storage required. Certain formats allow this container to be of different size: For example RGB 8:8:8 can have size of 3 and/or 4. Often, the 4 byte version is referred to as ARGB, but TSI does not process the Alpha (“A”) channel, so the presence of that is ignored.

6.1.4 TSI_REF1_ELEMENT_WIDTH

```
TSI_REF1_ELEMENT_WIDTH                         0x13
(TSI_REF1_PIXELS_PER_ELEMENT                   0x13)
unsigned int Ref1ElementW                      U32
4 bytes                                        RW
```

Description

Defines the width of a single element as number of pixels.

Important: *TSI_REF1_PIXELS_PER_ELEMENT name define is considered obsolete, however it continues to be defined for backwards compatibility. The new name was incorporated as it is more descriptive.*

6.1.5 TSI_REF1_ELEMENT_HEIGHT

```
TSI_REF1_ELEMENT_HEIGHT                        0x14
(TSI_REF1_LINES_PER_ELEMENT                    0x14)
unsigned int Ref1ElementH                      U32
4 bytes                                        RW
```

Description

Defines the height of a single element as number of pixels.

Important: *TSI_REF1_LINES_PER_ELEMENT name define is considered obsolete, however it continues to be defined for backwards compatibility. The new name was incorporated as it is more descriptive.*

6.1.6 TSI_REF1_COLOR_DEPTH

```
TSI_REF1_COLOR_DEPTH                          0x15
unsigned itn Ref1ColorDepth                   U32
4 bytes                                       RW
```

Description

Defines the color depth of the image as number of bits per color channel regardless of the color format. Please notice that this information is not in fact used as operational value in TSI, but rather as a generic meta information.

6.1.7 TSI_REF1_ELEMENT_FORMAT

```
TSI_REF1_ELEMENT_FORMAT                       0x16
(TSI_REF1_PIXEL_FORMAT                        0x16)
unsigned int Ref1ElementFormat                U32
4 bytes                                       RW
```

Description

Defines the element format used to encode the pixel data of the bitmap. Please see table below for currently defined format ID values:

Define	ID	Description
TSI_ELF_RGB_080808	0	RGB color, max color depth 8 bits per channel. Encoded as 3 unsigned bytes or 4 unsigned bytes per element.
TSI_ELF_RGB_161616	1	RGB color, max color depth 16 bits per channel. Encoded as 3 unsigned shorts or 4 unsigned shorts per element.
TSI_ELF_YCbCr_080808	0x100	YCbCr color, max color depth 8 bits per channel. Encoded as 3 unsigned bytes or 4 unsigned bytes per element.
TSI_ELF_YCbCr_161616	0x101	YCbCr color, max color depth 16 bits per channel. Encoded as 3 unsigned shorts or 4 unsigned shorts per element.

Important: *TSI_REF1_PIXEL_FORMAT name define is considered obsolete, however it continues to be defined for backwards compatibility. The new name was incorporated as it is more descriptive.*

6.1.8 TSI_REF1_FRAME_DATA

```
TSI_REF1_FRAME_DATA                           0x17
unsigned char Ref1Data[]                      ARRAY_U8
Variable size                                 RW
```

Description

Contains bitmap data encoded as defined in other TSI_REF1_* CI's.

6.1.9 TSI_TEST_LENGTH

```
TSI_TEST_LENGTH                                 0x1000
unsigned int TSI_VidTestLen                     U32
4 bytes                                         RW
```

Description

Defines the length of the video test as number of frames. Default setting is 60 frames.

Important: *32-bit version of TSI with very high resolution inputs may require the test length to be reduced.*

6.1.10 TSI_LIM_FRAME_MISMATCHES

```
TSI_LIM_FRAME_MISMATCHES                        0x1001
unsigned int TSI_FrameMismatches                U32
4 bytes                                         RW
```

Description

Defines number of frame that are allowed to be considered as “failed” before the entire test is considered as “failed”. Default setting is 0.

6.1.11 TSI_LIM_PIXEL_MISMATCHES

```
TSI_LIM_PIXEL_MISMATCHES                        0x1002
unsigned int TSI_PxlMismatches                  U32
4 bytes                                         RW
```

Description

Defines the number of pixels that allowed to be considered as “failed” before the frame is considered as “failed”. Default setting is 0.

6.1.12 TSI_PIXEL_TOLERANCE

```
TSI_PIXEL_TOLERANCE                             0x1003
unsigned int TSI_ColorTolerance                 U32
4 bytes                                         RW
```

Description

Defines maximum difference allowed between reference image and captured image. If the difference is larger than the value of this CI on any color channel, the pixel is considered “failed”. Default setting is 0.

6.1.13 TSI_MAX_AUTO_SAVE_FAILED

TSI_MAX_AUTO_SAVE_FAILED	0x1080
unsigned int TSI_MaxAutosave	U32
4 bytes	RW

Description

Maximum number of frames failed frames saved per test run. Default setting is 0. If the setting is “0”, no frames are saved.

6.1.14 TSI_FAILED_FRAME_TARGET_FOLDER

TSI_FAILED_FRAME_TARGET_FOLDER	0x1081
char FailedFramesFolder[]	ARRAY_U8
Variable size from 1 to 260 bytes	RW

Description

Contains the full path to the folder where failed frames are to be saved **without** trailing backslash (‘\’). No default. Failed frame file-name will be “Failed_<#>.ppm”, where <#> is replaced with an auto-incremented number.

6.1.15 TSI_MAX_EXPORT_FAILED

TSI_MAX_EXPORT_FAILED	0x1082
unsigned int MaxExportFailed	U32
4 bytes	RW

Description

Defines the number of failed frames to be exported from the video test. Default setting is 0. If the setting is 0, no frames are exported.

6.1.16 TSI_R_VIDEO_TEST_RAW_RESULTS_DATA

```
TSI_R_VIDEO_TEST_RAW_RESULTS_DATA                0x1008
unsigned int RawResults[]                        ARRAY_U32
Variable size                                    RO
```

Description

Provides access to an array of integers that contain error counts per each compared frame. The maximum size for this block is test length multiplied with size of 4 unsigned integers (unsigned 32 bit values). See below for description of each block of 4 integers:

Byte Offset	Description
0	Number of errors on red color channel (Or Cr channel for YCbCr).
4	Number of errors on green color channel (or Y channel for YCbCr).
8	Number of errors on blue color channel (or Cb channel for YCbCr).
12	Number of failed pixels.

6.1.17 TSI_EXPORTED

```
TSI_EXPORTED                                     0x108e
unsigned int Exported                            U32
4 bytes                                          RO
```

Description

Defines number of frames failed frames exported from the video test. No default. Please notice that exported frames are not accessible after the next test is started.

6.1.18 TSI_EXPORT_ACCESS_INDEX

```
TSI_EXPORT_ACCESS_INDEX                          0x108f
unsigned int ExportIndex                         U32
4 bytes                                          WO
```

Description

Defines which export frame information to access with CI ID's 0x1090 to 0x1097. Allowed value range is from 0 to number of exported frames.

6.1.19 TSI_EXPORT_WIDTH

```
TSI_EXPORT_WIDTH                                0x1090
unsigned int ExportW                            U32
4 bytes                                         RO
```

Description

Defines the width of an exported frame as number of elements.

6.1.20 TSI_EXPORT_HEIGHT

```
TSI_EXPORT_HEIGHT                               0x1091
unsinged int ExportH                            U32
4 bytes                                         RO
```

Description

Defines height of an exported frame as number of elements.

6.1.21 TSI_EXPORT_ELEMENT_SIZE

```
TSI_EXPORT_ELEMENT_SIZE                         0x1092
unsigned int ExportElemSize                     U32
4 bytes                                         RO
```

Description

Defines the size of element as number of bytes.

6.1.22 TSI_EXPORT_ELEMENT_WIDTH

```
TSI_EXPORT_ELEMENT_WIDHT                        0x1093
(TSI_EXPORT_PIXELS_PER_ELEMENT                  0x1093)
unsinged int ExportElemW                        U32
4 bytes                                         RO
```

Description

Defines the width of a single element in pixels.

Important: *TSI_EXPORT_PIXELS_PER_ELEMENT name define is considered obsolete, however it continues to be defined for backwards compatibility. The new name was incorporated as it is more descriptive.*

6.1.23 TSI_EXPORT_ELEMENT_HEIGHT

```
TSI_EXPORT_ELEMENT_HEIGHT                    0x1094
(TSI_EXPORT_LINES_PER_ELEMENT                0x1094)
unsigned int ExportElemH                     U32
4 bytes                                      RO
```

Description

Defines the height of a single element in pixels.

Important: *TSI_EXPORT_LINES_PER_ELEMENT name define is considered obsolete, however it continues to be defined for backwards compatibility. The new name was incorporated as it is more descriptive.*

6.1.24 TSI_EXPORT_COLOR_DEPTH

```
TSI_EXPORT_COLOR_DEPTH                       0x1095
unsinged int ExportColorDepth                U32
4 bytes                                      RO
```

Description

Defines the color depth per color channel as number of bits.

6.1.25 TSI_EXPORT_ELEMENT_FORMAT

```
TSI_EXPORT_ELEMENT_FORMAT                    0x1096
(TSI_EXPORT_PIXEL_FORMAT                     0x1096)
unsigned int ExportElemFormat                U32
4 bytes                                      RO
```

Description

Defines the element format used to encode the pixel data of the bitmap. Please see table below for currently defined format ID values:

Define	ID	Description
TSI_ELF_RGB_080808	0	RGB color, max color depth 8 bits per channel. Encoded as 3 unsigned bytes or 4 unsigned bytes per element.
TSI_ELF_RGB_161616	1	RGB color, max color depth 16 bits per channel. Encoded as 3 unsigned shorts or 4 unsigned shorts per element.
TSI_ELF_YCbCr_080808	0x100	YCbCr color, max color depth 8 bits per channel. Encoded as 3 unsigned bytes or 4 unsigned bytes per element.
TSI_ELF_YCbCr_161616	0x101	YCbCr color, max color depth 16 bits per channel. Encoded as 3 unsigned shorts or 4 unsigned shorts per element.

Important: *TSI_EXPORT_PIXEL_FORMAT name define is considered obsolete, however it continues to be defined for backwards compatibility. The new name was incorporated as it is more descriptive.*

6.1.26 TSI_EXPORT_FRAME_DATA

```
TSI_EXPORT_FRAME_DATA                          0x1097
void *ExportFrameData                          ARRAY_U8
Variable size                                  RO
```

Description

Contains the frame data. The data size (in bytes) can be calculated by multiplying TSI_EXPORT_WIDTH by TSI_EXPORT_HEIGHT by TSI_EXPORT_ELEMENT_SIZE.

6.2 CRC based video tests

ClientVersion 8, and higher *TSI Advanced license required*

6.2.1 CRC based Video Test set / CRC based single frame reference video test

```
#define TSI_TEST_DP_VIDEO_CRC_SINGLE_REF
    0x00060000
#define TSI_TEST_HD_VIDEO_CRC_SINGLE_REF
    0x000b0000
```

Synopsis

The test checks input frames to match with provided resolution and color depth, and the contents of the frames are checked to be identical with the reference through comparing CRC values of the reference frame and the input frame. This test uses only the first reference CRC value set.

Important: *The 0x00060000 is specific to DP sink ports, and the 0x000b0000 ID is specific to HDMI sink ports. Attempting to use DP ID with HDMI port will cause test run to fail with TSI_ERROR_UNSUPPORTED TEST.*

Configuration items

Define	Config ID	Default	Description	Reference
TSI_CRC_TIMEOUT	0x10300	1000	CRC Test max. run time, in milliseconds.	6.2.5
TSI_CRC_FRAMES_TO_TEST	0x10301	20	Number of input frames to be tested.	6.2.6
TSI_CRC_LIM_FRAME_MISMATCHES	0x10302	0	Number of frames that are allowed to have mismatching CRC.	6.2.7
TSI_CRC_REF_WIDTH	0x10303	1920	Required Input video width in pixels.	6.2.8
TSI_CRC_REF_HEIGHT	0x10304	1080	Required Input video height in pixels.	6.2.9
TSI_CRC_REF_COLORDEPTH	0x10305	24	Required bits per pixel on input video.	6.2.10
TSI_CRC_REFERENCE_CRC_VALUES	0x10306	-	Block of memory containing max. 65535 CRC value sets.	6.2.11
TSI_CRC_REQUIRED_FRAME_RATE	0x10307	0	Required input video frame rate, in millihertz (mHz).	6.2.12
TSI_CRC_FRAME_RATE_TOLERANCE	0x10308	0	Allowed deviation from require frame rate, in millihertz (mHz).	6.2.13
TSI_CRC_MOTION_TEST_ITERATIONS	0x10309	1	Number of iterations the defined CRC sequencey must be found.	6.2.14
TSI_CRC_COLOR_FORMAT	0x1030a	0	Color format. 0 = RGB. Other values invalid	6.2.15

6.2.2 CRC based Video Test set / CRC based single frame video stability test

```
#define TSI_TEST_DP_CRC_VIDEO_STABILITY       0x00060001
#define TSI_TEST_HD_CRC_VIDEO_STABILITY       0x000b0001
```

Synopsis

A simple test that is used to verify if a video stream is stable without providing a CRC value set as reference. If the CRC values remain identical for the duration of the test, the test is passed.

Important: *The 0x00060001 is specific to DP sink ports, and the 0x000b0001 ID is specific to HDMI sink ports. Attempting to use DP ID with HDMI port will cause test run to fail with TSI_ERROR_UNSUPPORTED TEST.*

Configuration items

Define	Config ID	Default	Description	Reference
TSI_CRC_TIMEOUT	0x10300	1000	CRC Test max. run time, in milliseconds.	6.2.5
TSI_CRC_FRAMES_TO_TEST	0x10301	20	Number of input frames to be tested.	6.2.6

6.2.3 CRC based Video Test set / CRC based sequence of frames reference video test

```
#define TSI_TEST_DP_CRC_VIDEO_SEQUENCE        0x00060002
#define TSI_TEST_HD_CRC_VIDEO_SEQUENCE        0x000b0002
```

Synopsis

The Source DUT should be sending a repeating video sequence to the TE, without the sequence containing any identical frames within the loop.

The test will first synchronize with the provided CRC sequence by finding the video frame with a CRC matching the CRC of the first frame in the reference sequence. Once a match is detected, the test proceeds comparing CRC values of every frame to the CRC values in the reference sequence. If the test fails to synchronize to the input video stream the test will fail with timeout. Test will fail immediately if a CRC mismatch is detected. DUT will PASS the test if TE finds input video resolution and color format matching to reference parameters, found reference frame sequence and no mismatches CRC in frame sequence.

Important: *The 0x00060002 is specific to DP sink ports, and the 0x000b0002 ID is specific to HDMI sink ports. Attempting to use DP ID with HDMI port will cause test run to fail with TSI_ERROR_UNSUPPORTED TEST.*

Configuration items

Define	Config ID	Default	Description	Reference
TSI_CRC_TIMEOUT	0x10300	1000	CRC Test max. run time, in milliseconds.	6.2.5
TSI_CRC_FRAMES_TO_TEST	0x10301	20	Number of input frames to be tested.	6.2.6
TSI_CRC_LIM_FRAME_MISMATCHES	0x10302	0	Number of frames that are allowed to have mismatching CRC.	6.2.7
TSI_CRC_REF_WIDTH	0x10303	1920	Required Input video width in pixels.	6.2.8
TSI_CRC_REF_HEIGHT	0x10304	1080	Required Input video height in pixels.	6.2.9
TSI_CRC_REF_COLORDEPTH	0x10305	24	Required bits per pixel on input video.	6.2.10
TSI_CRC_REFERENCE_CRC_VALUES	0x10306	-	Block of memory containing max. 65535 CRC value sets.	6.2.11
TSI_CRC_REQUIRED_FRAME_RATE	0x10307	0	Required input video frame rate, in millihertz (mHz).	6.2.12
TSI_CRC_FRAME_RATE_TOLERANCE	0x10308	0	Allowed deviation from require frame rate, in millihertz (mHz).	6.2.13
TSI_CRC_MOTION_TEST_ITERATIONS	0x10309	1	Number of iterations the defined CRC sequencey must be found.	6.2.14
TSI_CRC_COLOR_FORMAT	0x1030a	0	Color format. 0 = RGB. Other values invalid	6.2.15

6.2.4 CRC Based Video Test Set / CRC based continuous sequence of reference frames

```
#define TSI_TEST_DP_CRC_CONT_VIDEO_SEQUENCE   0x00060003
#define TSI_TEST_HD_CRC_CONT_VIDEO_SEQEUNCE   0x000b0003
```

Synopsis

The Source DUT should be sending a repeating video sequence to the TE, without the sequence containing any identical frames within the loop.

The test will first synchronize with the provided CRC sequence by finding the video frame with a CRC matching the CRC of the first frame in the reference sequence. Once the match is detected, the test proceeds comparing CRC values of every frame to the CRC values in the reference sequence. When the entire reference sequence has been compared, the test expects to find the same sequence repeated without any intermediate frames between the last defined frame and the first one. The reference sequence is tested multiple times, as defined by the TSI_CRC_MOTION_TEST_ITERATIONS configuration item. If the test fails to synchronize to the input video stream the test will fail with timeout. Test will fail immediately if a CRC mismatch is detected. DUT will PASS the test if TE finds input video resolution and color format matching the reference parameters, found reference frame sequence and no mismatches CRC in frame sequence.

Important: *The 0x00060003 is specific to DP sink ports, and the 0x000b0003 ID is specific to HDMI sink ports. Attempting to use DP ID with HDMI port will cause test run to fail with TSI_ERROR_UNSUPPORTED TEST.*

Configuration items

Define	Config ID	Default	Description	Reference
TSI_CRC_TIMEOUT	0x10300	1000	CRC Test max. run time, in milliseconds.	6.2.5
TSI_CRC_FRAMES_TO_TEST	0x10301	20	Number of input frames to be tested.	6.2.6
TSI_CRC_LIM_FRAME_MISMATCHES	0x10302	0	Number of frames that are allowed to have mismatching CRC.	6.2.7
TSI_CRC_REF_WIDTH	0x10303	1920	Required Input video width in pixels.	6.2.8
TSI_CRC_REF_HEIGHT	0x10304	1080	Required Input video height in pixels.	6.2.9
TSI_CRC_REF_COLORDEPTH	0x10305	24	Required bits per pixel on input video.	6.2.10
TSI_CRC_REFERENCE_CRC_VALUES	0x10306	-	Block of memory containing max. 65535 CRC value sets.	6.2.11
TSI_CRC_REQUIRED_FRAME_RATE	0x10307	0	Required input video frame rate, in millihertz (mHz).	6.2.12
TSI_CRC_FRAME_RATE_TOLERANCE	0x10308	0	Allowed deviation from require frame rate, in millihertz (mHz).	6.2.13
TSI_CRC_MOTION_TEST_ITERATIONS	0x10309	1	Number of iterations the defined CRC sequencey must be found.	6.2.14
TSI_CRC_COLOR_FORMAT	0x1030a	0	Color format. 0 = RGB. Other values invalid	6.2.15

6.2.5 TSI_CRC_TIMEOUT

```
TSI_CRC_TIMEOUT                                 0x10300
unsigned int crc_test_timeout                   U32
4 bytes                                         RW
```

Synopsis

Defines timeout for all CRC based video tests, in milliseconds. Default setting is 1000ms.

6.2.6 TSI_CRC_FRAMES_TO_TEST

```
TSI_CRC_FRAMES_TO_TEST                          0x10301
unsigned int crc_frames_to_test                 U32
4 bytes                                         RW
```

Synopsis

Defines timeout for all CRC based video tests, as number of frames. This setting, and the TSI_CRC_TIMEOUT CI together define the length of the test: The limit that is reached first applies. Set this value to zero to disable frame count limit.

6.2.7 TSI_CRC_LIM_FRAME_MISMATCHES

```
TSI_CRC_LIM_FRAME_MISMATCHES                    0x10302
unsigned int crc_mismatches_allowed             U32
4 bytes                                         RW
```

Synopsis

Defines number of frames that are allowed to fail without causing the test result to be “failed”. Default setting is 0.

6.2.8 TSI_CRC_REF_WIDTH

```
TSI_CRC_REF_WIDTH                               0x10303
unsigned int crc_ref_w                          U32
4 bytes                                         RW
```

Synopsis

Defined the expected video width, in pixels. If the video being received does not match this setting, the test will fail. Default setting is 1920.

6.2.9 TSI_CRC_REF_HEIGHT

```
TSI_CRC_REF_HEIGHT                          0x10304
unsigned int crc_ref_h                      U32
4 bytes                                     RW
```

Synopsis

Defines the expected video height, in pixels. If the video being received does not match this setting, the test will fail. Default setting is 1080.

6.2.10 TSI_CRC_REF_COLORDEPTH

```
TSI_CRC_REF_COLORDEPTH                      0x10305
unsigned int crc_ref_colordepth             U32
4 bytes                                     RW
```

Synopsis

Defines the color depth as bits per pixel. If the input video color depth does not match this setting, the test will fail. Default setting is 24.

6.2.11 TSI_CRC_REFERENCE_CRC_VALUES

```
TSI_CRC_REFERENCE_CRC_VALUES                0x10306
unsigned short CRC_Values[]                 ARRAY_U16
Variable size                               RW
```

Synopsis

Contains CRC reference values. Each CRC set consists of 3 16-bit words; One word for each color channel. Red / Cr color channel CRC is at the lowest address (first word), followed by Green / Y channel (second word) and then Blue / Cb channel (third word). Maximum number of CRC value sets is 65535. Default CRC set is empty (=no default value).

6.2.12 TSI_CRC_REQUIRED_FRAME_RATE

```
TSI_CRC_REQUIRED_FRAME_RATE                 0x10307
unsigned int crc_req_frate                  U32
4 bytes                                     RW
```

Synopsis

Defines the required frame rate for CRC based tests, in millihertz. Setting of zero (0) disables the frame-rate requirement. Default setting is 0.

6.2.13 TSI_CRC_FRAME_RATE_TOLERANCE

```
TSI_CRC_FRAME_RATE_TOLERANCE                    0x10308
unsigned int crc_frate_tolerance                U32
4 bytes                                         RW
```

Synopsis

Defines the maximum allowed deviation of input frame-rate from the required frame rate (TSI_CRC_REQUIRED_FRAME_RATE), in millihertz. When this setting is non-zero, it defines the range of allowed frame rate as requirements ± tolerance. If the frame-rate requirement is set to zero, this setting has no effect. Default setting is 50 mHz.

6.2.14 TSI_CRC_MOTION_TEST_ITERATIONS

```
TSI_CRC_MOTION_TEST_ITERATION                   0x10309
unsigned int crc_motion_test_iters              U32
4 bytes                                         RW
```

Synopsis

Defines the number of iterations the defined CRC sequnce must be found in order to pass the test. Default is 1.

6.2.15 TSI_CRC_COLOR_FORMAT

```
TSI_CRC_COLOR_FORMAT                            0x1030a
unsinged int crc_color_format                   U32
4 bytes                                         RW
```

Synopsis

Defines the color format of the expected video input. Default value is 0 (=RGB).

Important: *This configuration item is reserved for future support of additional color spaces. Currently, it must be set to zero.*

6.3 Validate audio signal frequency and glitch-free audio reproduction

ClientVersion 4, and later *Basic license required*

```
#define TSI_TEST_AUDIO_KILOHERTZ        3     // TEST ID
```

Synopsis

Perform frequency check on the digital audio content and verify the content to be glitch-free. This test assumes that a pure sine-wave audio signal content is being transmitted to the test equipment.

The test will first capture minimum of one second of audio content. The audio is then analyzed in two stages. First, the power spectrum is calculated and the highest peak must be within the defined window. The peak frequency check resolution is better than ±1 Hz. In second stage, the audio is checked to contain no random glitches, such as dropped or duplicated samples. This is achieved by examining how the RDV ("Relative Distortion Value") changes over time within the sampled audio.

The test is considered passed if the audio content spectrum has the highest power within the defined window, and the number of detected audio glitches does not exceed programmed value.

Configuration items

Define	Config ID	Default	Description	Reference
TSI_EXPECTED_SAMPLE_RATE	0x2020	44100	Expected sampling rate of audio signal	6.3.1
TSI_EXPECTED_AUDIO_FREQUENCY	0x2021	1000	Expected audible (sine) frequency as Hz	6.3.2
TSI_AUDIO_FREQUENCY_TOLERANCE	0x2022	1	Allowed deviation from expected frequency as Hz	6.3.3
TSI_AUDIO_GLITCH_DETECT_TRESHOLD	0x2023	327680 (= 5.0)	Glitch detect treshold value as fixed point number.	6.3.4
TSI_AUDIO_GLITCHES_ALLOWED	0x2024	0	Number of audio glitches allowed per test	6.3.5

(Continued...)

(...Continued)

Log messages

The audio test run is divided into four (4) stages. Stage one is test initialization, basic parameter validation and resource allocation.

```
Starting audio test (Test ID 3)
Stage 1: Test initialization. Test params:
  - Test length 65536 samples (1.49 seconds of audio)
  - Channel count = 2
  - Expected sample Rate = 44100
  - Reference Frequency = 1000 Hz
```

Important: *Test length (samples) is automatically selected to hold at least one second of audio for all channels.*

Important: *While it is possible to change the reference frequency, it is recommended to use the default frequency of 1kHz.*

Stage two is data gathering. During this stage, audio signal is captured to system memory.

Stage three is audio content frequency verification. The audio content must have the highest power peak within <Reference Frequency> ± <frequency tolerance> range:

```
Stage 2 Completed -- Entering Stage 3: Frequency check
  - Channel 0, Max power found at 999.95 Hz
  - Channel 1, Max power found at 999.95 Hz
```

Important: *The measurement accuracy is always better than ±0.5 Hz for pure sine signal.*

Stage four is audio glitch detection. The intent is to find frequently and randomly dropped, duplicated or otherwise damaged samples:

```
Stage 3 completed -- Entering Stage 4: Glitch detect
  - RDV value = 15.80
  - Glitch detected: Channel 0, Within sample range 3258 - 3386
    (RDV Value = 42.76)
  - Glitch detected: Channel 0, Within sample range 3302 - 3430
    (RDV Value = 42.19)
  - Glitch detected: Channel 0, Within sample range 7665 - 7793
    (RDV Value = 42.76)
```

The RDV (“Relative Distortion Value”) is calculated over the entire audio signal to provide a base-line RDV. The RDV value can vary greatly depending on how clean the audio signal is and can also be effected by the audio signal's amplitude, which is why the base line is calculated rather than programmed with strict limits. The ideal value for RDV is 1.00, but it is unreachable due to the limitations of digital audio and mathematical analysis.

Important: *The RDV is a unit-less value that comes out of a computational algorithm, and must be compared with other values that come out of the same algorithm with same expected input signal in order to draw conclusions – a single sample of RDV is useless.*

(Continued...)

(...Continued)

To detect a glitch, the calculated RDV must change more than allowed (<Base-line RDV ± <Audio Glitch detect threshold>). If a large enough change is detected, each detection is reported with information on which channel had it, range of samples within which it is located and the calculated RDV value for that range. A single glitch can be detected multiple times depending on the magnitude of the glitch.

6.3.1 TSI_EXPECTED_SAMPLE_RATE

```
TSI_EXPECTED_SAMPLE_RATE                    0x2020
unsigned int AudioSampleRate                U32
4 bytes                                     RW
```

Description

Sample rate that should be present when running the test, in Hz. Default setting is 44100 Hz. If the audio stream sample rate does not match, the test will result fail.

6.3.2 TSI_EXPECTED_AUDIO_FREQUENCY

```
TSI_EXPECTED_AUDIO_FREQUENCY                0x2021
unsinged int AudioFreq                      U32
4 bytes                                     RW
```

Description

Expected audible signal frequency that should be present when running the test, in Hz. Default setting is 1000 Hz.

6.3.3 TSI_AUDIO_FREQUENCY_TOLERANCE

```
TSI_AUDIO_FREQUENCY_TOLERANCE                0x2022
unsigned int AudioFreqTolerance              U32
4 bytes                                      RW
```

Description

Maximum allowed frequency deviation for the audible signal from the reference frequency, in Hz. Default setting is 1 Hz.

6.3.4 TSI_AUDIO_GLITCH_DETECT_TRESHOLD

```
TSI_AUDIO_GLITCH_DETECT_TRESHOLD             0x2023
unsigned int AudioGlitchTreshold             U32
4 bytes                                      RW
```

Description

This value defines the accepted RDV range by adding/subtracting it from the calculated base RDV when performing glitch detection. Lower values mean more sensitive to glitches – please note that setting this value too low will cause even perfectly good signal to fail the test. Valid range for this setting is 0 to 32767.0; The default setting is 5.0 (327680 scaled).

Important: *FIXED POINT ENCODING. When setting this value parameter, the value being set must be multiplied by 65536 and set as a 32-bit integer. When reading the value, the received value must be divided by 65536 and shown as a floating point quantity.*

6.3.5 TSI_AUDIO_GLITCHES_ALLOWED

```
TSI_AUDIO_GLITCHES_ALLOWED                   0x2024
unsigned int AudioMaxGlitches                U32
4 bytes                                      RW
```

Description

Defines how many glitches are allowed before the audio test is considered failed. Default setting is 0.

Important: *Due to implementation specific characteristics, a single (but very audible) glitch is probably detected multiple times. The number of times a glitch is detected depends greatly on the severity of the glitch, and it's location respective to the sine waveform. Because of this, setting a non-zero but very low value may not make sense.*

6.4 HDMI Electrical tests

ClientVersion 7, and higher TSI Electrical tests license required

6.4.1 Electrical Test Set / Power test

```
#define TSI_TEST_HDMI_EL_POWER_LINE          0x00020000
```

Synopsis

This test checks voltage level on the +5V power line of the DUT source. HDMI defines 4.7V … 5.3V as acceptable voltage range on the sink side connector. (Called “TP2” in the HDMI specification).

The test will measure the power line voltage with 0 mA load, and with 55 mA load as required in the CTS spceification (Test ID 7-11: +5V Power). The test will fail if voltage level on the power line is below or above the defined voltage range.

Configuration items

Define	Config ID	Default	Description	Reference
TSI_HDMI_RX_TIMEOUT	0x10200	5000	Electrical test timeout, (ms).	6.4.5
TSI_HDMI_RX_POWER_LOW_LIMIT	0x10201	4700	HDMI power-supply low limit, (mV).	6.4.6
TSI_HDMI_RX_POWER_HIGH_LIMIT	0x10202	5300	HDMI power-supply high limit, (mV).	6.4.7

6.4.2 Electrical Test Set / HPD test

```
#define TSI_TEST_HDMI_EL_HPD_LINE            0x00020002
```

Synopsis

HPD line test checks cable/DUT source HPD line for short circuits to power or ground.

The test runs in two stages:

1. The HPD line is released to logical high state and voltage is measured from the HPD line. If the voltage level on the HPD line is outside the defined HPD ONE voltage window, the test considers that the HPD line is shorted to ground or power depending if the measured value is below the allowed window, or above it.

2. The HPD line is driven to logical low state and voltage is measured from the HPD line. If the voltage level on the HPD line is outside the defined HPD ZERO voltage window, the test considers that the HPD line is shorted to ground or power depending if the measured value is below the allowed window, or above it.

(Continued...)

(...Continued)

Configuration items

Define	Config ID	Default	Description	Reference
TSI_HDMI_RX_TIMEOUT	0x10200	5000	Electrical test timeout, (ms).	6.4.5
TSI_HDMI_RX_HPD_ZERO_LOW_LIMIT	0x10205	0	HDMI HPD logical zero voltage level low limit, (mV).	6.4.10
TSI_HDMI_RX_HPD_ZERO_HIGH_LIMIT	0x10206	400	HDMI HPD logical zero voltage level high limit, (mV).	6.4.11
TSI_HDMI_RX_HPD_ONE_LOW_LIMIT	0x10207	2400	HDMI HPD logical one voltage level low limit, (mV).	6.4.12
TSI_HDMI_RX_HPD_ONE_HIGHT_LIMIT	0x10208	5300	HDMI HPD logical one voltage level high limit, (mV).	6.4.13

6.4.3 Electrical Test Set / DDC and CEC test

```
#define TSI_TEST_HDMI_EL_DDC_CEC_LINES        0x00020003
```

Synopsis

DDC/CEC lines test measured voltage from the SCL, SDA and CEC lines when not being driven low.

If the DDC or CEC line voltage levels are outside the defined ranges, the test fails.

Configuration items

Define	Config ID	Default	Description	Reference
TSI_HDMI_RX_TIMEOUT	0x10200	5000	Electrical test timeout, (ms).	6.4.5
TSI_HDMI_RX_DDC_LOW_LIMIT	0x10209	4500	HDMI DDC voltage level low limit, (mV).	6.4.14
TSI_HDMI_RX_DDC_HIGH_LIMIT	0x1020a	5500	HDMI DDC voltage level high limit, (mV).	6.4.15
TSI_HDMI_RX_CEC_ZERO_LOW_LIMIT	0x1020b	0	HDMI CEC Logical zero low voltage limit, (mV).	6.4.16
TSI_HDMI_RX_CEC_ZERO_HIGH_LIMIT	0x1020c	600	HDMI CEC logical zero hight voltage limit, (mV).	6.4.17
TSI_HDMI_RX_CEC_ONE_LOW_LIMIT	0x1020d	2500	HDMI CEC logical one low voltage limit, (mV).	6.4.18
TSI_HDMI_RX_CEC_ONE_HIGH_LIMIT	0x1020e	3600	HDMI CEC logical one high voltage limit, (mV).	6.4.19

(Continued...)

(...Continued)

6.4.4 Electrical Test Set / TMDS test

```
#define TSI_TEST_HDMI_EL_TMDS_LINES          0x00020001
```

Synopsis

This test measures average voltage levels on TMDS signal lines.

TMDS will guarantee DC balanced signalling. Sink will pull up a line to 3.3V AVcc voltage and source will pull down the line. On active HDMI line average voltage level is expected to fall below AVcc for the value of the voltage swing divide by two and defaults to range 2.6V… 3.1V. Values out of the set range mean a problem with TMDS lines, such as short circuit or broken output driver. An open circuit measures 3.3V AVcc. DVI TMDS test has the same functionality as HDMI, but voltage range defaults to 3.0V…3.1V. TMDS differential pair positive and negative lines are measured separately.

Important: *Acceptable range should be set by the user depending on the source DUT and cable setup.*

Configuration items

Define	Config ID	Default	Description	Reference
TSI_HDMI_RX_TIMEOUT	0x10200	5000	Electrical test timeout, (ms).	6.4.5
TSI_HDMI_RX_LINK_LOW_LIMIT	0x10203	2900	HDMI Link voltage low limit, (mV).	6.4.8
TSI_HDMI_RX_LINK_HIGH_LIMIT	0x10204	3100	HDMI Link voltage high limit, (mV).	6.4.9

6.4.5 TSI_HDMI_RX_TIMEOUT

```
TSI_HDMI_RX_TIMEOUT                          0x10200
unsigned int hdmi_rx_timeout                 U32
4 bytes                                      RW
```

Synopsis

Timeout period used for all HDMI RX electrical tests, in milliseconds. Default timeout is 5000ms.

6.4.6 TSI_HDMI_RX_POWER_LOW_LIMIT

```
TSI_HDMI_RX_POWER_LOW_LIMIT                    0x10201
unsigned int hdmi_rx_pwr_ll                    U32
4 bytes                                        RW
```

Synopsis

HDMI power line voltage low limit, in millivolts. The voltage detected from HDMI power line must be higher than this value in order to pass tests. Default setting is 4700mV.

6.4.7 TSI_HDMI_RX_POWER_HIGH_LIMIT

```
TSI_HDMI_RX_POWER_HIGH_LIMIT                   0x10202
unsigned int hdmi_rx_pwr_hl                    U32
4 bytes                                        RW
```

Synopsis

HDMI power line voltage high limit, in millivolts. The voltage detected from HDMI power line must be less than this value in order to pass tests. Default setting is 5300mV.

6.4.8 TSI_HDMI_RX_LINK_LOW_LIMIT

```
TSI_HDMI_RX_LINK_LOW_LIMIT                     0x10203
unsigned int hdmi_rx_lnk_ll                    U32
4 bytes                                        RW
```

Synopsis

HDMI link line voltage low limit, in millivolts. The voltage detected from HDMI link line(s) during test must be higher than this value in order to pass test. Default setting is 2900mV.

Important: *The acceptable setting for this value can be different for different types of DUT's. Proper calibration of this value will require testing multiple DUT's of same type in order to find typical value for the DUT in question.*

6.4.9 TSI_HDMI_RX_LINK_HIGH_LIMIT

```
TSI_HDMI_RX_LINK_HIGH_LIMIT                    0x10204
unsigned int hdmi_rx_lnk_hl                    U32
4 bytes                                        RW
```

Synopsis

HDMI link line voltage high limit, in millivolts. The voltage detected from HDMI link line(s) during test must be less than this value in order to pass test. Default setting is 3100mV.

Important: *The acceptable setting for this value can be different for different types of DUT's. Proper calibration of this value will require testing multiple DUT's of same type in order to find typical value for the DUT in question.*

6.4.10 TSI_HDMI_RX_HPD_ZERO_LOW_LIMIT

```
TSI_HDMI_RX_HPD_ZERO_LOW_LIMIT                 0x10205
unsigned int hdmi_rx_hpd_0_ll                  U32
4 bytes                                        RW
```

Synopsis

HDMI HPD logical zero voltage level, lower limit, in millivolts. When HPD line is expected to be in logical zero state, the measured voltage must be higher than this value in order to pass test. Default setting is 0mV.

6.4.11 TSI_HDMI_RX_HPD_ZERO_HIGH_LIMIT

```
TSI_HDMI_RX_HPD_ZERO_HIGH_LIMIT                0x10206
unsigned int hdmi_rx_hpd_0_hl                  U32
4 bytes                                        RW
```

Synopsis

HDMI HPD logical zero voltage level, higher limit, in millivolts. When HDP line is expected to be in logical zero state, the measured voltage must be lower than this value in order to pass test. Default setting is 400mV.

6.4.12 TSI_HDMI_RX_HPD_ONE_LOW_LIMIT

```
TSI_HDMI_RX_HPD_ONE_LOW_LIMIT                    0x10207
unsigned int hdmi_rx_hpd_1_ll                    U32
4 bytes                                          RW
```

Synopsis

HDMI HPD logical one voltage level, lower limit, in millivolts. When HPD line is expected to be in logical one state, the measured voltage must be less than this value in order to pass test. Default setting is 2400mV.

6.4.13 TSI_HDMI_RX_HPD_ONE_HIGH_LIMIT

```
TSI_HDMI_RX_HPD_ONE_HIGH_LIMIT                   0x10208
unsigned int hdmi_rx_hpd_1_hl                    U32
4 bytes                                          RW
```

Synopsis

HDMI HPD logical one voltage level, higher limit, in millivolts. When HPD line is expected to be in logical one state, the measured voltage must be less than this value in order to pass test. Default setting is 5300mV.

6.4.14 TSI_HDMI_RX_DDC_LOW_LIMIT

```
TSI_HDMI_RX_DDC_LOW_LIMIT                        0x10209
unsigned int hdmi_rx_ddc_ll                      U32
4 bytes                                          RW
```

Synopsis

DDC Line voltage low limit, in millivolts. Test will measure DDC line voltage when the line is not being driven low. The measured value must be higher than this value in order to pass test. Default setting is 4500mV.

6.4.15 TSI_HDMI_RX_DDC_HIGH_LIMIT

```
TSI_HDMI_RX_DDC_HIGH_LIMIT                       0x1020a
unsigned int hdmi_rx_ddc_hl                      U32
4 bytes                                          RW
```

Synopsis

DDC Line voltage high limit, in millivolts. Test will measure DDC line voltage when the line is not being driven low. The measured value must be lower than this value in order to pass test. Default setting is 5500mV.

6.4.16 TSI_HDMI_RX_CEC_ZERO_LOW_LIMIT

```
TSI_HDMI_RX_CEC_ZERO_LOW_LIMIT                0x1020b
unsigned int hdmi_rx_cec_0_ll                 U32
4 bytes                                       RW
```

Synopsis

CCE Line logical zero voltage level, lower limit, in millivolts. The CCE line voltage is measured when CCE line state is logical zero. The measured value must be higher than this value in order to pass test. Default setting is 0mV.

6.4.17 TSI_HDMI_RX_CEC_ZERO_HIGH_LIMIT

```
TSI_HDMI_RX_CEC_ZERO_HIGH_LIMIT               0x1020c
unsigned int hdmi_rx_cec_0_hl                 U32
4 bytes                                       RW
```

Synopsis

CCE Line logical zero voltage level, higher limit, in millivolts. The CCE line voltage is measured when CCE line state is logical zero. The measured value must be lower than this value in order to pass test. Default setting is 600mV.

6.4.18 TSI_HDMI_RX_CEC_ONE_LOW_LIMIT

```
TSI_HDMI_RX_CEC_ONE_LOW_LIMIT                 0x1020d
unsigned int hdmi_rx_cec_1_ll                 U32
4 bytes                                       RW
```

Synopsis

CCE Line logical one voltage level, lower limit, in millivolts. The CCE line voltage is measured when CCE line state is logical one. The measured value must be higher than this setting in order to pass test. Default setting is 2500mV.

6.4.19 TSI_HDMI_RX_CEC_ONE_HIGH_LIMIT

```
TSI_HDMI_RX_CEC_ONE_HIGH_LIMIT                0x1020e
unsigned int hdmi_rx_cec_1_hl                 U32
4 bytes                                       RW
```

Synopsis

CCE Line logical one voltage level, higher limit, in millivolts. The CCE line voltage is measured when CCE line state is logical one. The measured value must be lower that this setting in order to pass test. Default setting is 3600mV.

6.5 DP Electrical tests

ClientVersion 7, and higher TSI Electrical tests license required

6.5.1 Electical Test Set / Main Link test

```
#define TSI_TEST_DP_EL_MAIN_LINK              0x00010001
```

Synopsis

The test measures power of DP input signal and checks that the result lies within an allowed voltage window.

The measured value follows the input signal's amplitude and is large for large input swing. Measured power value depends on signal waveform and it varies because of e.g. used cable. Due to this, the measurement only provides a relative value which does not represent any absolute value, e.g. input signal voltage levels.

“No signal” -level is initially set to 2.3V. Note that even a disconnected line will give a relatively high value. Good signal levels are expected to be within range 2.6V…4.0V. The allowed voltage window should be set separately for each device model after testing of several units.

Measured values are expected to be close to each other within a differential pair. Also, all main link differential pair measurements should produce a value close to each other if link training result is the same for all pairs.

Measurement results are given in volts but this is only the voltage level of power measurement circuitry output and does not relate to input signal. Main link differential pair positive and negative lines are measured separately.

Configuration items

Define	Config ID	Default	Description	Reference
TSI_DP_RX_TEST_TIMEOUT	0x10100	5000	Electrical test timeout (ms).	6.5.4
TSI_DP_RX_LINKS_LOW_VOLTAGE	0x10101	2600	Data links low voltage (mV).	6.5.5
TSI_DP_RX_LINKS_HI_VOLTAGE	0x10102	4000	Data links hi voltage (mV).	6.5.5
TSI_DP_RX_MAX_DUT_LANE_COUNT	0x1010f	4	DUT Max. lanes.	6.5.13
TSI_DP_RX_MAX_DUT_LINK_RATE	0x10110	20	DUT Max. lane frequency as multiplier of 0.27Gbps	6.5.14

6.5.2 Electrical Test Set / AUX test

```
#define TSI_TEST_DP_EL_AUX_LINE                0x00010002
```

Synopsis

Verifies voltage levels on AUX lines, and AUX connectivity to DUT.

The test runs in two stages:

1. The idle AUX voltage level is measured. It is expected that voltages match to values defined by resistor dividers set by connected DisplayPort sink and source devices (see AUX CH Differential Pair in the DP specification).
2. The TE creates a short HPD pulse to have the DUT to generate an AUX request. The DUT is expected to read 0x200 – 0x205 DPCD registers. Test captures sync sequence of AUX transaction and checks the unit interval timings.

Configuration items

Define	Config ID	Default	Description	Reference
TSI_DP_RX_TEST_TIMEOUT	0x10100	5000	Electrical test timeout (ms).	6.5.4
TSI_DP_RX_AUX_P_IDLE_LOW_VOLTAGE	0x10107	20	AUX P-Line idle state low voltage limit (mV).	6.5.8
TSI_DP_RX_AUX_P_IDLE_HI_VOLTAGE	0x10108	500	AUX P-Line idle state hi voltage limit (mV).	6.5.8
TSI_DP_RX_AUX_N_IDLE_LOW_VOLTAGE	0x10109	2600	AUX N-Line idle state low voltage limit (mV).	6.5.9
TSI_DP_RX_AUX_N_IDLE_HI_VOLTAGE	0x1010a	3600	AUX N-Line idel state hi voltage limit (mV).	6.5.9
TSI_DP_RX_AUX_P_TRIG_VOLTAGE	0x1010b	150	AUX P-Line trigger voltage level (mV).	6.5.10
TSI_DP_RX_AUX_N_TRIG_VOLTAGE	0x1010c	200	AUX N-Line trigger voltage level (mV).	6.5.10
TSI_DP_RX_AUX_SIGNAL_CAPT_TIMEOUT	0x1010d	200	AUX Signal capture timeout (ms)	6.5.11
TSI_DP_RX_AUX_SIGNAL_CAPT_TRIES	0x1010e	5	AUX Signal capture retries.	6.5.12

6.5.3 Electrical Test Set / HPD test

```
#define TSI_TEST_DP_EL_HPD_LINE                0x00010000
```

Synopsis

HPD line test checks cable/DUT source HPD line for short circuits to power or ground.

The test runs in two stages:

1. The HPD line is released to logical high state and voltage is measured from the HPD line. If the voltage level on the HPD line is outside the defined HPD ONE voltage window, the test considers that the HPD line is shorted to ground or power depending if the measured value is below the allowed window, or above it.
2. The HPD line is driven to logical low state and voltage is measured from the HPD line. If the voltage level on the HPD line is outside the defined HPD ZERO voltage window, the test considers that the HPD line is shorted to ground or power depending if the measured value is below the allowed window, or above it.

Configuration items

Define	Config ID	Default	Description	Reference
TSI_DP_RX_TEST_TIMEOUT	0x10100	5000	Electrical test timeout (ms).	6.5.4
TSI_DP_RX_HPD_ZERO_LOW_VOLTAGE	0x10103	-100	HPD logical zero voltage level low limit (mV).	6.5.6
TSI_DP_RX_HPD_ZERO_HI_VOLTAGE	0x10104	800	HPD logical zero voltage level hi limit (mV).	6.5.6
TSI_DP_RX_HPD_ONE_LOW_VOLTAGE	0x10105	800	HPD logical one voltage level low limit (mV).	6.5.7
TSI_DP_RX_HPD_ONE_HI_VOLTAGE	0x10106	5500	HDP logical one voltage level hi limit (mV).	6.5.7

6.5.4 TSI_DP_RX_TEST_TIMEOUT

```
TSI_DP_RX_TEST_TIMEOUT                          0x10100
unsigned int dp_rx_test_timeout                 U32
4 bytes                                         RW
```

Synopsis

Timeout period used for all DP RX electrical tests, in milliseconds. Default timeout is 5000ms.

6.5.5 TSI_DP_RX_LINKS_*_VOLTAGE

```
TSI_DP_RX_LINKS_LOW_VOLTAGE                    0x10101
TSI_DP_RX_LINKS_HI_VOLTAGE                     0x10102
unsigned int dp_rx_link_ll, dp_rx_link_hl      U32
4 bytes                                        RW
```

Synopsis

These two CI's define the acceptable voltage range DP link lines. The measured voltage must be higher than TSI_DP_RX_LINKS_LOW_VOLTAGE setting, and lower than TSI_DP_RX_LINKS_HI_VOLTAGE setting in order to pass test. Default setting for low voltage limit is 2600mV, and for high voltage limit 4000mV.

6.5.6 TSI_DP_RX_HPD_ZERO_*_VOLTAGE

```
TSI_DP_RX_HPD_ZERO_LOW_VOLTAGE                 0x10103
TSI_DP_RX_HDP_ZERO_HI_VOLTAGE                  0x10104
unsigned int dp_rx_hpd_0_ll, dp_rx_hpd_0_hl    U32
4 bytes                                        RW
```

Synopsis

These to CI's define the acceptable voltage range for HDP line when it is in logical zero state. The measured voltage must be higher than TSI_DP_RX_HDP_ZERO_LOW_VOLTAGE setting, and lower than TSI_DP_RX_HPD_ZERO_HI_VOLTAGE setting in order to pass test. Default setting for low voltage limit is -100mV, and for high voltage limit 800mV.

6.5.7 TSI_DP_RX_HDP_ONE_*_VOLTAGE

```
TSI_DP_RX_HDP_ONE_LOW_VOLTAGE                  0x10105
TSI_DP_RX_HDP_ONE_HI_VOLTAGE                   0x10106
unsigned int dp_rx_hpd_1_ll, dp_rx_hpd_1_hl    U32
4 bytes                                        RW
```

Synopsis

These two CI's define the acceptable voltage range for HPD line when it is in logical one state. The measured voltage must be higher than TSI_DP_RX_HDP_ONE_LOW_VOLTAGE setting, and lower than TSI_DP_RX_HDP_ONE_HI_VOLTAGE setting in order to pass test. Default setting for low voltage limit is 800mV, and for high voltage limit 5500mV.

6.5.8 TSI_DP_RX_AUX_P_IDLE_*_VOLTAGE

```
TSI_DP_RX_AUX_P_IDLE_LOW_VOLTAGE              0x10107
TSI_DP_RX_AUX_P_IDLE_HI_VOLTAGE               0x10108
unsigned int dp_rx_aux_p_ll, dp_rx_aux_p_hl   U32
4 bytes                                       RW
```

Synopsis

These two CI's define the acceptable AUX+ line idle voltage range when the AUX is idle. The measured voltage must be higher than TSI_DP_RX_AUX_P_IDLE_LOW_VOLTAGE setting, and lower than TSI_DP_RX_AUX_P_IDLE_HI_VOLTAGE setting in order to pass test. Default setting for low voltage limit is 500mV, and for high voltage limit 2600mV.

6.5.9 TSI_DP_RX_AUX_N_IDLE_*_VOLTAGE

```
TSI_DP_RX_AUX_N_IDLE_LOW_VOLTAGE              0x10109
TSI_DP_RX_AUX_N_IDLE_HI_VOLTAGE               0x1010a
unsigned int dp_rx_aux_n_ll, dp_rx_aux_n_hl   U32
4 bytes                                       RW
```

Synopsis

These two CI's defined the acceptable AUX- line idle voltage range when the AUX is idle. The measured voltage must be higher than TSI_DP_RX_AUX_N_IDLE_LOW_VOLTAGE setting, and lower than TSI_DP_RX_AUX_N_IDLE_HI_VOLTAGE setting in order to pass test. Default setting for low voltage limit is 2600mV, and for high voltage limit 3600mV.

6.5.10 TSI_DP_RX_AUX_*_TRIG_VOLTAGE

```
TSI_DP_RX_AUX_P_TRIG_VOLTAGE                  0x1010b
TSI_DP_RX_AUX_N_TRIG_VOLTAGE                  0x1010c
unsigned int dp_rx_aux_ptrig, dp_rx_aux_ntrig U32
4 bytes                                       RW
```

Synopsis

These two CI's define the AUX+ (TSI_DP_RX_AUX_P_TRIG_VOLTAGE) and AUX- (TSI_DP_RX_AUX_N_TRIG_VOLTAGE) line state change trigger levels. Default settings are for AUX+ 150mV and for AUX- 200mV.

6.5.11 TSI_DP_RX_AUX_SIGNAL_CAPT_TIMEOUT

```
TSI_DP_RX_AUX_SIGNAL_CAPT_TIMEOUT              0x1010d
unsigned int dp_rx_aux_cap_timeout             U32
4 bytes                                        RW
```

Synopsis

Timeout for AUX signal capture, in milliseconds. When the TE generates a HPD pulse during test, it waits for this amount of time (max.) for DUT to read DPCD locations 0x200 to 0x205. If this transaction is not seen, the test will fail. Default setting is 200ms.

6.5.12 TSI_DP_RX_AUX_SIGNAL_CAPT_TRIES

```
TSI_DP_RX_AUX_SIGNAL_CAPT_TRIES                0x1010e
unsigned int dp_rx_aux_cap_retries             U32
4 bytes                                        RW
```

Synopsis

Retry count for AUX signal capture. If the AUX signal capture after TE generated a HPD pulse fails, the TE will re-try this many times. Default setting is 5.

6.5.13 TSI_DP_RX_MAX_DUT_LANE_COUNT

```
TSI_DP_RX_MAX_DUT_LANE_COUNT                   0x1010f
unsigned int dp_rx_max_dut_lanes               U32
4 bytes                                        RW
```

Synopsis

Maximum number of lanes supported by the connected DUT. Typical values are 1, 2 or 4. Default setting is 4.

6.5.14 TSI_DP_RX_MAX_DUT_LINK_RATE

```
TSI_DP_RX_MAX_DUT_LINK_RATE                    0x10110
unsigned int dp_rx_max_dut_link_rate           U32
4 bytes                                        RW
```

Synopsis

Maximum link rate supported by the connected DUT, as multiplier of 0.27Gbps. Typical values are 6 (RBR), 10 (HBR), 20 (HBR-2) or 30 (HBR-3). Please note that HBR3 speed is not supported on all TE devices.

6.6 USB-C Electrical tests

ClientVersion 10, and higher Electrical test license required

6.6.1 USBC Electrical Test Set / Up Face port CC and Vconn test

```
#define TSI_TEST_USBC_CC_VCON                    0x000c0000
```

Synopsis

This test verifies operation of CC lines against short-circuit and open-circuit failures, and that directly related hardware is working properly. During the test, TE will operate as Type-C UFP device.

At test start, the TE will temporarily disconnect CC lines to simulate a re-plug event. After the re-plug event, Power source capability resistor R_a is connected CC2 line, and R_d resistor is connected to CC1 line. DUT is expected to have resistor R_p, or a current source applied to both CC1 and CC2 lines. The impedance of the DUT's R_p resistor, or current source must be adjusted so that the voltage drop on R_d resistor on the TE is within one of the voltage ranges defined by *TSI_USBC_EL_CC_LOW_VOLTAGE_** and *TSI_USBC_EL_CC_HI_VOLTAGE_**. (By default the ranges are 261mV → 588mV, 675mV → 1189mV and 1238mV → 2181mV). TE will measure the voltage drop on R_p.

The TE will measure the voltage present on CC2 after DUT has started to provide Vconn voltage on CC2 for an active cable.

Once the Vconn is measured, the TE will enable the cable-flip feature and repeat the steps as above.

In order to pass the test, the measured values from CC1, CC2 and Vconn must be within a respective range. The passable ranges are defined by the following configuration items:

- CC lines voltage range 1: *TSI_USBC_EL_CC_LOW_VOLTAGE_1* and *TSI_USBC_EL_CC_HI_VOLTAGE_1*.
- CC lines voltage range 2: *TSI_USBC_EL_CC_LOW_VOLTAGE_2* and *TSI_USBC_EL_CC_HI_VOLTAGE_2*.
- CC lines voltage range 3: *TSI_USBC_EL_CC_LOW_VOLTAGE_3* and *TSI_USBC_EL_CC_HI_VOLTAGE_3*.
- Vconn voltag range: *TSI_USBC_EL_VCON_LOW_VOLTAGE* and *TSI_USBC_EL_VCON_HI_VOLTAGE.*

These configuration items should be programmed with values averaged from several fully operational DUT's.

Important: *In order to run this test with UCD-340, a special cable provided by Unigraf must be used.*

(Continued...)

(...Continued)

Configuration items

Define	Config ID	Default	Description	Reference
TSI_USBC_EL_TIMEOUT	0x10500	5000	Test maximum runtime, in ms.	6.6.5
TSI_USBC_EL_REPLUG_TIME	0x10502	1500	Replug disconnected time, in ms.	6.6.7
TSI_USBC_EL_DUT_ATTACH_TIMEOUT	0x10503	10000	Time TE waits DUT to connect after cable plug, in ms.	6.6.8
TSI_USBC_EL_CC_LOW_VOLTAGE_1	0x10505	261	CC low limit for 0.5A/0.9A, in mV	6.6.10
TSI_USBC_EL_CC_HI_VOLTAGE_1	0x10506	588	CC hi limit for 0.5A/0.9A, in mV	6.6.11
TSI_USBC_EL_CC_LOW_VOLTAGE_2	0x10507	675	CC low limit for 1.5A, in mV	6.6.12
TSI_USBC_EL_CC_HI_VOLTAGE_2	0x10508	1189	CC hi limit for 1.5A, in mV	6.6.13
TSI_USBC_EL_CC_LOW_VOLTAGE_3	0x10509	1238	CC low limit for 3.0A, in mV	6.6.14
TSI_USBC_EL_CC_HI_VOLTAGE_3	0x1050a	2181	CC hi limit for 3.0A, in mV	6.6.15
TSI_USBC_EL_VCON_LOW_VOLTAGE	0x1050b	4750	Low limit for Vconn, in mV	6.6.16
TSI_USBC_EL_VCON_HI_VOLTAGE	0x1050c	5500	Hi limit for Vconn, in mV	6.6.17

RAW Test results data

Important: *The RAW test results data is available after running the test, starting a new test on the same device will invalidate the contents of the RAW test results.*

Define	Config ID	Default	Description	Reference
TSI_R_TDATA_USBC_EL_VCC1	0xf801	-	CC1 line voltage, in mV	
TSI_R_TDATA_USBC_EL_VCONN1	0xf802	-	CC2 line voltage when Vconn, in mV	
TSI_R_TDATA_USBC_EL_VCC2	0xf803	-	CC2 line voltage, in mV	
TSI_R_TDATA_USBC_EL_VCONN2	0xf804	-	CC2 line voltage when Vconn, in mV	

6.6.2 USBC Electrical Test Set / AUX (SBU) lines test

```
#define TSI_TEST_USBC_SBU_DP_AUX                0x000c0001
```

Synopsis

This test verifies operation of SBU lines against short-circuit and open-circuit failures, and that directly related hardware is working properly. During the test the TE will operate as Type-C UFP device. In order to run this test, the DUT must support DisplayPort alternate mode.

At test start, the TE will temporarily disconnect CC lines to simulate a re-plug event. After the re-plug event, the TE waits for the DUT to enter DP Alternate mode. Once the DUT has entered the DP alternate mode, TE will measure voltage levels on AUX+ (SBU1) line, and AUX- (SBU2) line. Please notice that if the TE is acting as DP Sink, it will de-assert the HPD signal to keep AUX bus at IDLE state during the voltage measurements.

Once the voltages are measured, the TE will enable the cable-flip feature and repeat the above steps.

In order to pass the test, all the measured AUX- and AUX+ voltages must be within the respective ranges (By default AUX- range is 100mV → 600mV, and AUX+ range is 2500mV → 3000mV). The ranges can be configured with the following configuration items:

- AUX+ voltage range: *TSI_USBC_AUX_P_IDLE_LOW_VOLTAGE* and *TSI_USBC_AUX_P_IDLE_HI_VOLTAGE*.
- AUX- voltage range: *TSI_USBC_AUX_N_IDLE_LOW_VOLTAGE* and *TSI_USBC_AUX_N_IDLE_HI_VOLTAGE.*

These configuration items should be programmed with values averaged from several fully operational DUT's.

Important: *In order to run this test with UCD-340, a special cable provided by Unigraf must be used.*

(Continued...)

Configuration items

Define	Config ID	Default	Description	Reference
TSI_USBC_EL_TIMEOUT	0x10500	5000	Test maximum runtime, in ms.	6.6.5
TSI_USBC_EL_DUT_CAPS	0x10501	0	Defines DUT caps. Bits	6.6.6
TSI_USBC_EL_REPLUG_TIME	0x10502	1500	Replug disconnected time, in ms.	6.6.7
TSI_USBC_EL_DUT_ATTACH_TIMEOUT	0x10503	10000	Time TE waits DUT to connect after cable plug, in ms.	6.6.8
TSI_USBC_EL_AUX_P_IDLE_LOW_VOLTAGE	0x1050e	100	DP AUX+ low voltage when idle, mV	6.6.19
TSI_USBC_EL_AUX_P_IDLE_HI_VOLTAGE	0x1050f	600	DP AUX+ hi voltage when idle, mV	6.6.20
TSI_USBC_EL_AUX_N_IDLE_LOW_VOLTAGE	0x10510	2500	DP AUX- low voltage when idle, mV	6.6.21
TSI_USBC_EL_AUX_N_IDLE_HI_VOLTAGE	0x10511	3000	DP AUX- hi voltage when idle, mV	6.6.22

RAW Test results data

Important: *The RAW test results data is available after running the test, starting a new test on the same device will invalidate the contents of the RAW test results.*

Define	Config ID	Default	Description	Reference
TSI_R_TDATA_USBC_VAUX1_P	0xf801	-		
TSI_R_TDATA_USBC_VAUX1_N	0xf802	-		
TSI_R_TDATA_USBC_VAUX2_P	0xf803	-		
TSI_R_TDATA_USBC_VAUX2_N	0xf804	-		

6.6.3 USBC Electrical Test Set / DUT as Power Sink

```
#define TSI_TEST_USBC_DUT_PWR_SINK            0x000c0002
```

Synopsis

This test verifies operation Vbus and GND lines for short-circuit and open-circuit failures. The test is performed using mandatory PDO for power contract. During the test, the TE will operate as power source, and advertise only the mandatory PDO for power contract. In order to run this test, the DUT must support Power Sink role.

The test starts by negotiating the power contract. Once the power contract is established, the TE will wait for power measurement delay before measuring current over Vbus and GND lines. Voltage over Vbus is also measured. The power measurement delay can be modified with the *TSI_USBC_EL_PWR_MEASURE_DELAY* Configuration item. The purpose of the delay is to allow the DUT some time to stabilize it's power consumption. As the currents are measured one at a time, any variance in power consumption in the DUT during the measurement can cause this test to fail. The test assumes that the current flows through the four separate Vbus and GND lines evenly, and the contacts are verified with this characteristic in mind. Total currents are calculated for Vbus, and for GND. The highest difference between each of the four connections may not exceed the programmed deviation limits. The deviation is defined as per-mill of the total currents for Vbus and GND respectively.

In order to pass the test, the measured Vbus voltage must be between the *TSI_USBC_EL_VBUS_LOW_VOLTAGE* and *TSI_USBC_EL_VBUS_HI_VOLTAGE*. In addition, the currents measured from Vbus may not deviate more than indicated by *TSI_USBC_EL_VBUS_CURRENT_MAX_DEV*. Also, the currents measured from GND lines may not deviate more than indicated by *TSI_USBC_EL_GND_CURRENT_MAX_DEV*.

Important: *In order to run this test with UCD-340, a special cable provided by Unigraf must be used.*

Important: *In order to run this test with UCD-340, the Electrical Testing add-on board must be installed on the device.*

(Continued...)

(...Continued)

Configuration items

Define	Config ID	Default	Description	Reference
TSI_USBC_EL_TIMEOUT	0x10500	5000	Test maximum runtime, in ms.	6.6.5
TSI_USBC_EL_DUT_CAPS	0x10501	0	Defines DUT caps. Bits	6.6.6
TSI_USBC_EL_REPLUG_TIME	0x10502	1500	Replug disconnected time, in ms.	6.6.7
TSI_USBC_EL_DUT_ATTACH_TIMEOUT	0x10503	10000	Time TE waits DUT to connect after cable plug, in ms.	6.6.8
TSI_USBC_EL_PWR_CONTRACT_TIMEOUT	0x10504	5000	Time TE waits DUT to complete power contract, in ms.	6.6.9
TSI_USBC_EL_VBUS_LOW_VOLTAGE	0x10512	4750	Vbus low voltage limit, mV	6.6.23
TSI_USBC_EL_VBUS_HI_VOLTAGE	0x10513	5500	Vbus hi voltage limit, mV	6.6.24
TSI_USBC_EL_VBUS_CURRENT_MAX_DEV	0x10514	100	Highest allowed deviation between the Vbus pins, as ‰ (1/1000) parts of measured current	6.6.25
TSI_USBC_EL_GND_CURRENT_MAX_DEV	0x10515	100	Highest allowed deviationb between the GND pins as ‰ (1/1000) parts of measured current	6.6.26
TSI_USBC_EL_PWR_MEASURE_DELAY	0x10516	2000	Delay from power contract to current measurements, ms	6.6.27
TSI_USBC_EL_MIN_DUT_CURRENT	0x10517	0	Minimum current the DUT must draw , mA. (0 = Disabled)	6.6.28
TSI_USBC_EL_DELAY_AFTER_LOAD	0x10518	0	Delay after applying load resistors, ms.	6.6.29

RAW Test results data

Important: *The RAW test results data is available after running the test, starting a new test on the same device will invalidate the contents of the RAW test results.*

Define	Config ID	Default	Description	Reference
TSI_R_TDATA_USBC_EL_VBUS_V	0xf801	-	Vbus voltage, in mV.	
TSI_R_TDATA_USBC_EL_VBUS_I1	0xf802	-	Vbus current, line 1, in mA.	
TSI_R_TDATA_USBC_EL_VBUS_I2	0xf803	-	Vbus current, line 2, in mA.	
TSI_R_TDATA_USBC_EL_VBUS_I3	0xf804	-	Vbus current, line 3, in mA.	
TSI_R_TDATA_USBC_EL_VBUS_I4	0xf805	-	Vbus current, line 4, in mA.	
TSI_R_TDATA_USBC_EL_GND_I1	0xf806	-	Gnd current, line 1, in mA.	
TSI_R_TDATA_USBC_EL_GND_I2	0xf807	-	Gnd current, line 2, in mA.	
TSI_R_TDATA_USBC_EL_GND_I3	0xf808	-	Gnd current, line 3, in mA.	
TSI_R_TDATA_USBC_EL_GND_I4	0xf809	-	Gnd current, line 4, in mA.	

6.6.4 USBC Electrical Test Set / DUT as Power Source

```
#define TSI_TEST_USBC_DUT_PWR_SOURCE          0x000c0003
```

Synopsis

This test verifies operation Vbus and GND lines for short-circuit and open-circuit failures. The test is performed using mandatory PDO for power contract. During the test, the TE will operate as power sink, and selects the mandatory PDO for power contract. In order to run this test, the DUT must support Power Source role.

The test starts by negotiating the power contract. Once the power contract is established, the TE will wait for power measurement delay before measuring current over Vbus and GND lines. Voltage over Vbus is also measured. The power measurement delay can be modified with the *TSI_USBC_EL_PWR_MEASURE_DELAY* Configuration item. As the currents are measured one at a time, any variance in power delivery form the DUT during the measurement can cause this test to fail. The test assumes that the current flows through the four separate Vbus and GND lines evenly, and the contacts are verified with this characteristic in mind. Total currents are calculated for Vbus, and for GND. The highest difference between each of the four connections may not exceed the programmed deviation limits. The deviation is defined as per-mill of the total currents for Vbus and GND respectively.

In order to pass the test, the measured Vbus voltage must be between the *TSI_USBC_EL_VBUS_LOW_VOLTAGE* and *TSI_USBC_EL_VBUS_HI_VOLTAGE*. In addition, the currents measured from Vbus may not deviate more than indicated by *TSI_USBC_EL_VBUS_CURRENT_MAX_DEV*. Also, the currents measured from GND lines may not deviate more than indicated by *TSI_USBC_EL_GND_CURRENT_MAX_DEV*.

Important: *In order to run this test with UCD-340, a special cable provided by Unigraf must be used.*

Important: *In order to run this test with UCD-340, the Electrical Testing add-on board must be installed on the device.*

(Continued...)

(...Continued)

Configuration items

Define	Config ID	Default	Description	Reference
TSI_USBC_EL_TIMEOUT	0x10500	5000	Test maximum runtime, in ms.	6.6.5
TSI_USBC_EL_DUT_CAPS	0x10501	0	Defines DUT caps. Bits	6.6.6
TSI_USBC_EL_REPLUG_TIME	0x10502	1500	Replug disconnected time, in ms.	6.6.7
TSI_USBC_EL_DUT_ATTACH_TIMEOUT	0x10503	10000	Time TE waits DUT to connect after cable plug, in ms.	6.6.8
TSI_USBC_EL_PWR_CONTRACT_TIMEOUT	0x10504	5000	Time TE waits DUT to complete power contract, in ms.	6.6.9
TSI_USBC_EL_VBUS_LOW_VOLTAGE	0x10512	4750	Vbus low voltage limit, mV	6.6.23
TSI_USBC_EL_VBUS_HI_VOLTAGE	0x10513	5500	Vbus hi voltage limit, mV	6.6.24
TSI_USBC_EL_VBUS_CURRENT_MAX_DEV	0x10514	100	Highest allowed deviation between the Vbus pins, as ‰ (1/1000) parts of measured current	6.6.25
TSI_USBC_EL_GND_CURRENT_MAX_DEV	0x10515	100	Highest allowed deviationb between the GND pins as ‰ (1/1000) parts of measured current	6.6.26
TSI_USBC_EL_PWR_MEASURE_DELAY	0x10516	2000	Delay from power contract to current measurements, ms	6.6.27
TSI_USBC_EL_MIN_DUT_CURRENT	0x10517	0	Minimum current the DUT must draw , mA. (0 = Disabled)	6.6.28
TSI_USBC_EL_DELAY_AFTER_LOAD	0x10518	0	Delay after applying load resistors, ms.	6.6.29

RAW Test results data

Important: *The RAW test results data is available after running the test, starting a new test on the same device will invalidate the contents of the RAW test results.*

Define	Config ID	Default	Description	Reference
TSI_R_TDATA_USBC_EL_VBUS_V	0xf801	-	Vbus voltage, in mV.	
TSI_R_TDATA_USBC_EL_VBUS_I1	0xf802	-	Vbus current, line 1, in mA.	
TSI_R_TDATA_USBC_EL_VBUS_I2	0xf803	-	Vbus current, line 2, in mA.	
TSI_R_TDATA_USBC_EL_VBUS_I3	0xf804	-	Vbus current, line 3, in mA.	
TSI_R_TDATA_USBC_EL_VBUS_I4	0xf805	-	Vbus current, line 4, in mA.	
TSI_R_TDATA_USBC_EL_GND_I1	0xf806	-	Gnd current, line 1, in mA.	
TSI_R_TDATA_USBC_EL_GND_I2	0xf807	-	Gnd current, line 2, in mA.	
TSI_R_TDATA_USBC_EL_GND_I3	0xf808	-	Gnd current, line 3, in mA.	
TSI_R_TDATA_USBC_EL_GND_I4	0xf809	-	Gnd current, line 4, in mA.	

6.6.5 TSI_USBC_EL_TIMEOUT

```
TSI_USBC_EL_TIMEOUT                          0x00010500
unsigned int usvc_el_timeout                 U32
4 bytes                                      RW
```

Synopsis

Defines the test activity maximum run-time, in milliseconds. Default setting is 5000ms.

Important: *When the test is waiting for DUT with a max. delay this timeout is not advancing during the wait.*

6.6.6 TSI_USBC_EL_DUT_CAPS

```
TSI_USBC_EL_DUT_CAPS                         0x00010501
unsigned int usbc_el_dut_caps                U32
4 bytes                                      RW
```

Synopsis

Defines DUT capabilities. Please see flag definitions below. Default setting is 0.

Bits	Description	
0	0	DUT does not support DP Alt Mode.
	1	DUT does support DP Alt Mode.
1	0	DUT cannot be Power Source.
	1	DUT can be Power Source.
2	0	DUT cannot be Power Sink.
	1	DUT can be Power Sink.
3	0	DUT does support Power Delivery Contract.
	1	DUT does not support Power Delivery Contract.
31:4	RESERVED	

6.6.7 TSI_USBC_EL_REPLUG_TIME

```
TSI_USBC_EL_REPLUG_TIME                      0x00010502
unsigned int usbc_replug_time                U32
4 bytes                                      RW
```

Synopsis

Defines the time period for USB Type-C re-plug simulation “disconnected” state. The period is defined in milliseconds. Default value is 1500ms.

6.6.8 TSI_USBC_EL_DUT_ATTACH_TIMEOUT

```
TSI_USBC_EL_DUT_ATTACH_TIMEOUT                0x00010503
unsigned int usbc_dut_attach_timeout          U32
4 bytes                                       RW
```

Synopsis

Defines the time period that the TE will wait for DUT to complete connection after cable plug. Time is defined in milliseconds. Default value is 10000ms.

6.6.9 TSI_USBC_EL_PWR_CONTRACT_TIMEOUT

```
TSI_USBC_EL_PWR_CONTRACT_TIMEOUT              0x00010504
unsigned int usbc_pwr_contract_timeout        U32
4 bytes                                       RW
```

Synopsis

Defines the time period that the TE will wait for DUT to complete power contract negotiation. Time is defined in milliseconds. Default value is 5000ms.

6.6.10 TSI_USBC_EL_CC_LOW_VOLTAGE_1

```
TSI_USBC_EL_CC_LOW_VOLTAGE_1                  0x00010505
unsigned int usbc_cc_low_voltage1             U32
4 bytes                                       RW
```

Synopsis

Defines the low limit for the voltage window when power sink current is 0.5A or 0.9A. The limit is defined in millivolts (mV). Default setting is 261mV.

6.6.11 TSI_USBC_EL_CC_HI_VOLTAGE_1

```
TSI_USBC_EL_CC_HI_VOLTAGE_!                   0x00010506
unsigned int usbc_cc_hi_voltage1              U32
4 bytes                                       RW
```

Synopsis

Defines the high limit for the voltage window when power sink current is 0.5A or 0.9A. The limit is defined in millivolts (mV). Default settingf is 588mV.

6.6.12 TSI_USBC_EL_CC_LOW_VOLTAGE_2

```
TSI_USBC_EL_CC_LOW_VOLTAGE_2                    0x00010507
unsigned int usbc_cc_low_voltage2               U32
4 bytes                                         RW
```

Synopsis

Defines the low limit for the voltage window when power sink current is 1.5A. The limit is defined in millivolts (mV). Default setting is 675mV.

6.6.13 TSI_USBC_EL_CC_HI_VOLTAGE_2

```
TSI_USBC_EL_CC_HI_VOLTAGE_2                     0x00010508
unsigned int usbc_cc_hi_voltage2                U32
4 bytes                                         RW
```

Synopsis

Defines the high limit for the voltage window when power sink current is 1.5A. The limit is defined in millivolts (mV). Default setting is 1189mV.

6.6.14 TSI_USBC_EL_CC_LOW_VOLTAGE_3

```
TSI_USBC_EL_CC_LOW_VOLTAGE_3                    0x00010509
unsigned int usbc_cc_low_voltage3               U32
4 bytes                                         RW
```

Synopsis

Defines the low limit for the voltage window when power sink current is 3.0A. The limit is defined in millivolts (mV). Default setting is 1238mV.

6.6.15 TSI_USBC_EL_CC_HI_VOLTAGE_3

```
TSI_USBC_EL_HI_VOLTAGE_3                        0x0001050a
unsigned int usbc_cc_hi_voltage3                U32
4 bytes                                         RW
```

Synopsis

Defines the high limit for the voltage window when power sink current is 3.0A. The limit is defined in millivolts (mV). Default setting is 2181mV.

6.6.16 TSI_USBC_EL_VCON_LOW_VOLTAGE

```
TSI_USBC_EL_VCON_LOW_VOLTAGE                    0x0001050b
unsigned int usbc_vcon_low_voltage              U32
4 bytes                                         RW
```

Synopsis

Defines the low limit for the Vcon voltage window. The limit is defined in millivolts (mV). Default setting is 4750mV.

6.6.17 TSI_USBC_EL_VCON_HI_VOLTAGE

```
TSI_USBC_EL_VCON_HI_VOLTAGE                     0x0001050c
unsigned int usbc_vcon_hi_voltage               U32
4 bytes                                         RW
```

Synopsis

Defines the high limit for the Vcon voltage window. The limit is defined in millivolts (mV). Default setting is 5500mV.

6.6.18 TSI_USBC_EL_DP_ALT_TIMEOUT

```
TSI_USBC_EL_DP_ALT_TIMEOUT                      0x0001050d
unsigned int usbc_dp_alt_timeout                U32
4 bytes                                         RW
```

Synopsis

Defines the timeout the TE will wait for the DUT to enter into DisplayPort alternate mode. The timeout is defined in milliseconds. Default setting is 5000ms.

6.6.19 TSI_USBC_EL_AUX_P_IDLE_LOW_VOLTAGE

```
TSI_USBC_EL_AUX_P_IDLE_LOW_VOLTAGE              0x0001050e
unsigned int usbc_aux_p_idle_lo_voltage         U32
4 bytes                                         RW
```

Synopsis

Defines the low voltage limit for the positive DP AUX line when idle. The limit is defined in millivolts (mV). Default setting 100mV.

6.6.20 TSI_USBC_EL_AUX_P_IDLE_HI_VOLTAGE

```
TSI_USBC_EL_AUX_P_IDLE_HI_VOLTAGE            0x0001050f
unsigned int usbc_aux_p_idle_hi_voltage      U32
4 bytes                                      RW
```

Synopsis

Defines the high voltage limit for the positive DP AUX line when idle. The limit is defined in millivolts (mV). Default setting is 600mV.

6.6.21 TSI_USBC_EL_AUX_N_IDLE_LOW_VOLTAGE

```
TSI_USBC_EL_AUX_N_IDLE_LOW_VOLTAGE           0x00010510
unsigned int usbc_aux_n_idle_lo_voltage      U32
4 bytes                                      RW
```

Synopsis

Defines the low voltage limit for the negative DP AUX line when idle. The limit is defined in millivolts (mV). Default setting is 2500mV.

6.6.22 TSI_USBC_EL_AUX_N_IDLE_HI_VOLTAGE

```
TSI_USBC_EL_AUX_N_IDLE_HI_VOLTAGE            0x00010511
unsigned int usbc_aux_n_idle_hi_voltage      U32
4 bytes                                      RW
```

Synopsis

Defines the high voltage limit for the negative DP AUX line when idle. The limit is defined in millivolts (mV). Default setting is 3000mV.

6.6.23 TSI_USBC_EL_VBUS_LOW_VOLTAGE

```
TSI_USBC_EL_VBUS_LOW_VOLTAGE                 0x00010512
unsigned int usbc_vbus_lo_voltage            U32
4 bytes                                      RW
```

Synopsis

Defines the low limit for Vbus voltage window. The limit is defined in millivolts (mV). Default setting is 4750mV.

6.6.24 TSI_USBC_EL_VBUS_HI_VOLTAGE

```
TSI_USBC_EL_VBUS_HI_VOLTAGE                    0x00010513
unsigned int usbc_vbus_hi_voltage              U32
4 bytes                                        RW
```

Synopsis

Defines the high limit for Vbus voltage window. The limit is defined in millivolts (mV). Default setting is 5500mV.

6.6.25 TSI_USBC_EL_VBUS_CURRENT_MAX_DEV

```
TSI_USBC_EL_VBUS_CURRENT_MAX_DEV               0x00010514
unsigned int vbus_max_deviation                U32
4 bytes                                        RW
```

Synopsis

Defines the highest allowed deviation between maximum and minimum currents measured from the individual Vbus pins as per-mill (‰) of total measured current. This means that if the total measured current is 3000mA, and the setting 100, the maximum difference that is allowed between maximum and minimum currents is 300mA. Default setting is 100‰.

6.6.26 TSI_USBC_EL_GND_CURRENT_MAX_DEV

```
TSI_USBC_EL_GND_CURRENT_MAX_DEV                0x00010515
unsigned int gnd_max_deviation                 U32
4 bytes                                        RW
```

Synopsis

Defines the highest allowed deviation between maximum and minimum currents measured from the individual GND pins as per-mill (‰) of total measured current. This means that if the total measured current is 3000mA, and the setting 100, the maximum difference that is allowed between maximum and minimum currents is 300mA. Default setting is 100‰.

6.6.27 TSI_USBC_EL_PWR_MEASURE_DELAY

```
TSI_USBC_EL_PWR_MEASURE_DELAY                  0x00010516
unsigned int pwr_measure_delay                 U32
4 bytes                                        RW
```

Synopsis

Defines delay from end of power contract negotiation to voltage / current measurements. The delay is defined in milliseconds, and the default setting is 2000ms

6.6.28 TSI_USBC_EL_MIN_DUT_CURRENT

```
TSI_USBC_EL_MIN_DUT_CURRENT                    0x00010517
unsigned int dut_min_power_use                 U32
4 bytes                                        RW
```

Synopsis

Defines the minimum current, in mA, that a Power Sink DUT must use in order to pass the test. Set this value to zero (0) to disable minimum current check. Default value is 0 (=Disabled).

6.6.29 TSI_USBC_EL_DELAY_AFTER_LOAD

```
TSI_USBC_EL_DELAY_AFTER_LOAD                   0x00010518
```

Synopsis

Defines delay from applying load resistors, in milliseconds. This setting is used to allow the power source to settle and reach stable operation before measurement. Default value is 0.

6.7 CEC functional Test set / CEC functional test

```
ClientVersion 7, and higher
```

```
#define TSI_TEST_HDMI_CEC                          0x00050000
```

Synopsis

The test verifies that source DUT correctly handles HPD event, reads EDID and broadcasts the CEC *"Report physcal address"* message.

First, the TE allocates the given physical address and issues a HPD pulse simulating cable detach/attach. The it waits for DUT to broadcast the CEC "Report physical address" message. The test is considered passed if the TE finds that the DUT broadcasts with the physical address allocated by the TE for the test.

Important: *As a side effect, the CEC will also verify functionality of HPD and EDID reading if the test passes.*

Configuration items

Define	Config ID	Default	Description	Reference
TSI_HDMI_RX_CEC_TIMEOUT	0x10400	5000	HDMI CEC functional test timeout, in milliseconds	6.7.1
TSI_HDMI_RX_CEC_LOCAL_PHY_ADDR	0x10401	0x4000	Local PHY Address.	6.7.2

6.7.1 TSI_HDMI_RX_CEC_TIMEOUT

```
TSI_HDMI_RX_CEC_TIMEOUT                          0x10400
unsigned int cec_timeout                         U32
4 bytes                                          RW
```

Synopsis

Defines the CEC functional test timeout, in milliseconds. The test must complete within this time-period in order to succeed. Default setting is 5000ms.

6.7.2 TSI_HDMI_RX_CEC_LOCAL_PHY_ADDR

```
TSI_HDMI_RX_CEC_LOCA_PHY_ADDR                    0x10401
unsigned int cec_phy_addr                        U32
4 bytes                                          RW
```

Synopsis

Defines the CEC local PHY address. The address is stored in lowest 16-bits. Default setting is 0x4000 (“4.0.0.0”).

```
Typically these addresses are given as “A.B.C.D”, similar to IP addresses.
Each number in the address can be a value between 0 and 15.
Therefore, address “8.9.10.11” would become HEX value 0x000089AB.
```

6.8 Link Test set / Link Training at All Supported Lane Counts and Link Rates

```
ClientVersion 8, and higher
```

```
#define TSI_TEST_DP_SIMPLE_LINK                0x00070000
```

Synopsis

Test requests link training on all supported lane counts and link rates. Each link training must be successfully completed in order to pass the test.

Configuration items

Define	Config ID	Default	Description	Reference
TSI_DP_LTT_TIMEOUT	0x10700	5000	Timeout for each test loop iteration, in milliseconds.	6.8.1
TSI_DP_LTT_MAX_LANE_COUNT	0x10701	4	Max. number of lanes to be tested. Valid settings are 1, 2 and 4	6.8.2
TSI_DP_LTT_MAX_RATE	0x10702	20	Max. link rate to be tested as multiple of 0.27 Gbps. Valid settings are 6, 10 and 20.	6.8.3
TSI_DP_LTT_HPD_PULSE_DURATION	0x10705	1000	HPD pulse duration used to trigger link-training, in milliseconds.	6.8.4
TSI_DP_LTT_LT_START_TIMEOUT	0x10706	5000	Timeout for link training start after HPD pulse, in milliseconds.	6.8.5
TSI_DP_LTT_TEST_LOOP_DELAY	0x10707	3000	Delay time after link training before the next next test loop iteration, in milliseconds.	6.8.6

6.8.1 TSI_DP_LTT_TIMEOUT

```
TSI_DP_LTT_TIMEOUT                                  0x10700
unsigned int ltt_timeout                            U32
4 bytes                                             RW
```

Synopsis

Defines timeout for each test iteration, in milliseconds. The test iterates through a number of iterations depending on other tests. Each iteration must complete within this timeout in order for the test succeed. Default setting is 5000ms.

6.8.2 TSI_DP_LTT_MAX_LANE_COUNT

```
TSI_DP_LTT_MAX_LANE_COUNT                    0x10701
unsigned int ltt_max_lanes                   U32
4 bytes                                      RW
```

Synopsis

Defines the maximum number of lanes to be tested. Valid settings are 1, 2 and 4. Default setting is 4.

6.8.3 TSI_DP_LTT_MAX_RATE

```
TSI_DP_LTT_MAX_RATE                          0x10702
unsigned int ltt_max_rate                    U32
4 bytes                                      RW
```

Synopsis

Defines the maximum link rate to be tested. The setting is in multiplier of 0.27Gbps. Valid settings are 6, 10 and 20. Default setting is 20.

6.8.4 TSI_DP_LTT_HPD_PULSE_DURATION

```
TSI_DP_LTT_HPD_PULSE_DURATION                0x10705
unsigned int ltt_hpd_duration                U32
4 bytes                                      RW
```

Synopsis

Defines the length of the HPD pulse used to start each test iteration, in milliseconds. Default setting is 1000ms.

6.8.5 TSI_DP_LTT_LT_START_TIMEOUT

```
TSI_DP_LTT_LT_START_TIMEOUT                  0x10706
unsigned int ltt_lt_start_timeout            U32
4 bytes                                      RW
```

Synopsis

Defines how long the test waits for LT start after issuing HPD pulse, in milliseconds. Default setting is 5000ms.

6.8.6 TSI_DP_LTT_TEST_LOOP_DELAY

```
TSI_DP_LTT_TEST_LOOP_DELAY                    0x10707
unsigned int ltt_loop_delay                   U32
4 bytes                                       RW
```

Synopsis

Defines the additional delay inserted in between test iterations, in milliseconds. Default setting is 3000ms.

6.9 DP 1.4 Link Layer CTS for Source DUT

ClientVersion 12, and higher *License: TBD*

This section defines the LL CTS tests for DP 1.4 Source DUT. Please note that unlike other tests, the operation of these tests are not described here. Please refer to the DisplayPort CTS standard for test operation descriptions.

Important: *All of the test ID values and CI ID value definitions in this section are to be considered subject for changes, as the LL CTS for DP 1.4 specifications are not yet officially completed!*

6.9.1 Test ID definitions

The following test ID values are defined:

Define	Test ID	DP 1.4 LL CTS standrad test.
TSI_DP14_LL_CTS_SRC_DUT_4_2_1_1	0x000e0000	4.2.1.1 Source DUT Retry on No-Reply During AUX Read after HPD Plug Event
TSI_DP14_LL_CTS_SRC_DUT_4_2_1_2	0x000e0001	4.2.1.2 Source Retry on Invalid Reply During AUX Read after HPD Plug Event
TSI_DP14_LL_CTS_SRC_DUT_4_2_2_1	0x000e0002	4.2.2.1 DPCD Receiver Capability and EDID Read upon HPD Plug Event
TSI_DP14_LL_CTS_SRC_DUT_4_2_2_2	0x000e0003	4.2.2.2 DPCD Receiver Capability Read upon HPD Plug Event
TSI_DP14_LL_CTS_SRC_DUT_4_2_2_3	0x000e0004	4.2.2.3 EDID Read
TSI_DP14_LL_CTS_SRC_DUT_4_2_2_4	0x000e0005	4.2.2.4 EDID Read Failure #1: I2C-Over-AUX NACK
TSI_DP14_LL_CTS_SRC_DUT_4_2_2_5	0x000e0006	4.2.2.5 EDID Read Failure #2: I2C-Over-AUX DEFER
TSI_DP14_LL_CTS_SRC_DUT_4_2_2_6	0x000e0007	4.2.2.6 EDID Corruption Detection
TSI_DP14_LL_CTS_SRC_DUT_4_2_2_7	0x000e0008	4.2.2.7 Branch Device Detection upon HPD Plug Event
TSI_DP14_LL_CTS_SRC_DUT_4_2_2_8	0x000e0009	4.2.2.8 EDID Read on IRQ HPD Event after Branch Device Detection
TSI_DP14_LL_CTS_SRC_DUT_4_2_2_9	0x000e000a	4.2.2.9 E-DDC Four Block EDID Read
TSI_DP14_LL_CTS_SRC_DUT_4_2_2_10	0x000e000b	4.2.2.10 Link Status-Adjust Request AUX read interval during Link Training
TSI_DP14_LL_CTS_SRC_DUT_4_3_1_1	0x000e000c	4.3.1.1 Successful LT at All Supported Lane Counts and Link Speeds
TSI_DP14_LL_CTS_SRC_DUT_4_3_1_2	0x000e000d	4.3.1.2 Successful Link Training Upon HPD Plug Event
TSI_DP14_LL_CTS_SRC_DUT_4_3_1_3	0x000e000e	4.3.1.3 Successful LT with Request of Higher Differential Voltage Swing During Clock Recovery Sequence
TSI_DP14_LL_CTS_SRC_DUT_4_3_1_4	0x000e000f	4.3.1.4 Successful LT to a Lower Link Rate #1: Iterate at Max Voltage Swing
TSI_DP14_LL_CTS_SRC_DUT_4_3_1_5	0x000e0010	4.3.1.5 Successful LT to a Lower Link Rate #2: Iterate at Minimum Voltage Swing
TSI_DP14_LL_CTS_SRC_DUT_4_3_1_6	0x000e0011	4.3.1.6 Successful LT with Request of a Higher Pre-emphasis Setting During Channel Equalization Sequence
TSI_DP14_LL_CTS_SRC_DUT_4_3_1_7	0x000e0012	4.3.1.7 Successful LT at Lower Link Rate Due to Loss of Symbol Lock During Channel Equalization Sequence

(Continued...)

(...Continued)

Define	Test ID	Description
TSI_DP14_LL_CTS_SRC_DUT_4_3_1_8	0x000e0013	4.3.1.8 Unsuccessful LT at Lower Link Rate #1: Iterate at Max Voltage Swing
TSI_DP14_LL_CTS_SRC_DUT_4_3_1_9	0x000e0014	4.3.1.9 Unsuccessful LT at Lower Link Rate #2: Iterate at Minimum Voltage Swing
TSI_DP14_LL_CTS_SRC_DUT_4_3_1_10	0x000e0015	4.3.1.10 Unsuccessful LT due to Failure in Channel Equalization Sequence [loop count > 5]
TSI_DP14_LL_CTS_SRC_DUT_4_3_1_11	0x000e0016	4.3.1.11 Successful LT with Simultaneous Request for Differential Voltage Swing and Pre-emphasis during Clock Recovery Sequence
TSI_DP14_LL_CTS_SRC_DUT_4_3_2_1	0x000e0017	4.3.2.1 Successful Link Re-training After IRQ HPD Pulse Due to Loss of Symbol Lock
TSI_DP14_LL_CTS_SRC_DUT_4_3_2_2	0x000e0018	4.3.2.2 Successful Link Re-training After IRQ HPD Pulse Due to Loss of Clock Recovery Lock
TSI_DP14_LL_CTS_SRC_DUT_4_3_2_3	0x000e0019	4.3.2.3 Successful Link Re-training After IRQ HPD Pulse Due to Loss of Inter-lane Alignment Lock
TSI_DP14_LL_CTS_SRC_DUT_4_3_2_4	0x000e001a	4.3.2.4 Handling of IRQ HPD Pulse with No Error Status Bits Set
TSI_DP14_LL_CTS_SRC_DUT_4_3_2_5	0x000e001b	4.3.2.5 Lane Count Reduction
TSI_DP14_LL_CTS_SRC_DUT_4_3_3_1	0x000e001c	4.3.3.1 Video Time Stamp Generation
TSI_DP14_LL_CTS_SRC_DUT_4_4_1_1	0x000e001d	4.4.1.1 Data Packing and Steering
TSI_DP14_LL_CTS_SRC_DUT_4_4_1_2	0x000e001e	4.4.1.2 Main Stream Data Packing and Stuffing - Least Packed TU
TSI_DP14_LL_CTS_SRC_DUT_4_4_1_3	0x000e001f	4.4.1.3 Main Stream Data Packing and Stuffing - Most Packed TU
TSI_DP14_LL_CTS_SRC_DUT_4_4_2	0x000e0020	4.4.2 Main Video Stream Format Change Handling
TSI_DP14_LL_CTS_SRC_DUT_4_4_3	0x000e0021	4.4.3 Power Management
TSI_DP14_LL_CTS_SRC_DUT_400_1_1	0x000e0022	400.1.1 Source Device HPD Event Pulse Length Test
TSI_DP14_LL_CTS_SRC_DUT_400_1_2	0x000e0023	400.1.2 Source Device IRQ_HPD Pulse Length Test
TSI_DP14_LL_CTS_SRC_DUT_400_1_3	0x000e0024	400.1.3 Source Device Inactive HPD / Inactive AUX Test
TSI_DP14_LL_CTS_SRC_DUT_400_2_1	0x000e0025	400.2.1 Source Device Link Training CR Fallback Test
TSI_DP14_LL_CTS_SRC_DUT_400_2_2	0x000e0026	400.2.2 Source Device Link Training EQ Fallback Test

All tests are executed as defined in the DP 1.4 LL CTS specifications.

Important: *The test ID values are subject for changes due to the CTS specification not being officially ready.*

Configuration items

Define	Config ID	Default	Description	Reference
TSI_DP14_SRCCTS_TIMEOUT	0x010e0000	5000	Test timeout in milliseconds.	6.9.2
TSI_DP14_SRCCTS_MAX_LANES	0x010e0001	4	Maximum number of lanes supported by the DUT.	6.9.3
TSI_DP14_SRCCTS_MAX_LINK_RATE	0x010e0002	20	Maximum link rate supported by the DUT as multiplier for 0.27Gbps.	6.9.4
TSI_DP14_SRCCTS_DUT_CAPS	0x010e0003	0	DUT Capability flags	6.9.5
TSI_DP14_SRCCTS_DUT_TA	0x010e0004	0	DUT Test automation flags	6.9.6

(Continued...)

(...Continued)

Define	Config ID	Default	Description	Reference
TSI_DP14_SRCCTS_LONG_HPD_PULSE	0x010e0005	1000	Long HPD pulse duration, in milliseconds.	6.9.7
TSI_DP14_SRCCTS_LT_START_TIMEOUT	0x010e0006	5000	Link Training start timeout, in milliseconds.	6.9.8
TSI_DP14_SRCCTS_TEST_CYCLE_DELAY	0x010e0007	3000	Test cycle delay, in milliseconds	6.9.9
TSI_DP14_SRCCTS_COLOR_FORMATS	0x010e0008	2	Number of valid color format definitions.	6.9.10
TSI_DP14_SRCCTS_FAILSAFE_MODE	0x010e0009	1	Fail-safe video mode ID.	6.9.11
TSI_DP14_SRCCTS_MAX_RESOLUTION	0x010e000a	5	Maximum resolution supported by the DUT.	6.9.12
TSI_DP14_SRCCTS_1L_MOST_PACKED	0x010e0020	8	Most packed video timing definitions for 1, 2 and 4 lanes links.	6.9.13
TSI_DP14_SRCCTS_2L_MOST_PACKED	0x010e0021	12		
TSI_DP14_SRCCTS_4L_MOST_PACKED	0x010e0022	18		
TSI_DP14_SRCCTS_1L_RBB	0x010e0023	2	Video timings used for Reduced Bitrate (RBR, 1.62Gbps) testing for 1, 2 and 4 lanes.	6.9.14
TSI_DP14_SRCCTS_2L_RBB	0x010e0024	2		
TSI_DP14_SRCCTS_4L_RBB	0x010e0025	3		
TSI_DP14_SRCCTS_1L_HBR	0x010e0026	3	Video timings used for High Bitrate (HBR, 2.7Gbps) testing for 1, 2 and 4 lanes.	
TSI_DP14_SRCCTS_2L_HBR	0x010e0027	3		
TSI_DP14_SRCCTS_4L_HBR	0x010e0028	4		
TSI_DP14_SRCCTS_1L_HBR2	0x010e0029	4	Video timings used for High Bitrate-2 (HBR2, 5.4Gbps) testing for 1, 2 and 4 lanes.	
TSI_DP14_SRCCTS_2L_HBR2	0x010e002a	4		
TSI_DP14_SRCCTS_4L_HBR2	0x010e002b	5		
TSI_DP14_SRCCTS_1L_HBR3	0x010e002c	4	Video timings used for High Bitrate-3 (HBR3, 8.1Gbps) testing for 1, 2 and 4 lanes.	
TSI_DP14_SRCCTS_2L_HBR3	0x010e002d	6		
TSI_DP14_SRCCTS_4L_HBR3	0x010e002e	7		
TSI_DP14_SRCCTS_COLOR_MODE_0	0x010e0038	0	These parameters are considered mandatory and should not be changed.	6.9.15
TSI_DP14_SRCCTS_COLOR_MODE_1	0x010e0039	0x20		
TSI_DP14_SRCCTS_COLOR_MODE_2	0x010e003a	0	Optional and additional color modes to be used with DP CTS tests. The CI with ID value 0x010e0008 (DP14_SRCCTS_COLOR_FORMATS) should indicate the number of color modes defined in this table. Un-used color mdoes should be set to zero.	
TSI_DP14_SRCCTS_COLOR_MODE_3	0x010e003b	0		
TSI_DP14_SRCCTS_COLOR_MODE_4	0x010e003c	0		
TSI_DP14_SRCCTS_COLOR_MODE_5	0x010e003d	0		
TSI_DP14_SRCCTS_COLOR_MODE_6	0x010e003e	0		
TSI_DP14_SRCCTS_COLOR_MODE_7	0x010e003f	0		
TSI_DP14_SRCCTS_COLOR_MODE_8	0x010e0040	0		
TSI_DP14_SRCCTS_COLOR_MODE_9	0x010e0041	0		
TSI_DP14_SRCCTS_COLOR_MODE_10	0x010e0042	0		
TSI_DP14_SRCCTS_COLOR_MODE_11	0x010e0043	0		
TSI_DP14_SRCCTS_COLOR_MODE_12	0x010e0044	0		
TSI_DP14_SRCCTS_COLOR_MODE_13	0x010e0045	0		
TSI_DP14_SRCCTS_COLOR_MODE_14	0x010e0046	0		

(Continued...)

(...Continued)

Define	Config ID	Default	Description	Reference
TSI_DP14_SRCCTS_COLOR_MODE_15	0x010e0047	0	Optional and additional color modes to be used with DP CTS tests. The CI with ID value 0x010e0008 (DP14_SRCCTS_COLOR_FORMATS) should indicate the number of color modes defined in this table. Un-used color mdoes should be set to zero.	6.9.15
TSI_DP14_SRCCTS_COLOR_MODE_16	0x010e0048	0		
TSI_DP14_SRCCTS_COLOR_MODE_17	0x010e0049	0		
TSI_DP14_SRCCTS_COLOR_MODE_19	0x010e004b	0		
TSI_DP14_SRCCTS_COLOR_MODE_20	0x010e004c	0		
TSI_DP14_SRCCTS_COLOR_MODE_21	0x010e004d	0		
TSI_DP14_SRCCTS_COLOR_MODE_22	0x010e004e	0		
TSI_DP14_SRCCTS_COLOR_MODE_23	0x010e004f	0		
TSI_DP14_SRCCTS_COLOR_MODE_24	0x010e0050	0		
TSI_DP14_SRCCTS_COLOR_MODE_25	0x010e0051	0		
TSI_DP14_SRCCTS_COLOR_MODE_26	0x010e0052	0		
TSI_DP14_SRCCTS_COLOR_MODE_27	0x010e0053	0		
TSI_DP14_SRCCTS_COLOR_MODE_28	0x010e0054	0		
TSI_DP14_SRCCTS_COLOR_MODE_29	0x010e0055	0		
TSI_DP14_SRCCTS_COLOR_MODE_30	0x010e0056	0		
TSI_DP14_SRCCTS_COLOR_MODE_31	0x010e0057	0		
TSI_DP14_SRCCTS_MIN_SAMPLE_RATE	0x010e0010	44100	Minimum supported sample rate	
TSI_DP14_SRCCTS_MAX_SAMPLE_RATE	0x010e0011	44100	Maximum supported sample rate	
TSI_DP14_SRCCTS_CHANNELS1	0x010e0012	2	Minimum number of channels for minimum sample rate	
TSI_DP14_SRCCTS_CH_ALLOC1	0x010e0013	0	Channel allocation bits	
TSI_DP14_SRCCTS_CHANNELS2	0x010e0014	2	Maximum number of channels for minimum sample rate	
TSI_DP14_SRCCTS_CH_ALLOC2	0x010e0015	0	Channel allocation bits	
TSI_DP14_SRCCTS_CHANNELS3	0x010e0016	2	Minimum number of channels for maximum sample rate	
TSI_DP14_SRCCTS_CH_ALLOC3	0x010e0017	0	Channel allocations bits	
TSI_DP14_SRCCTS_CHANNELS4	0x010e0018	2	Maximum number of channels for maximum sample rate	
TSI_DP14_SRCCTS_CH_ALLOC4	0x010e0019	0	Channel allocation bits	
TSI_DP14_SRCCTS_AUDIO_TEST_PATTERN	0x010e001a	0	Audio test pattern	
TSI_DP14_SRCCTS_EDID_SAMPLE_SIZE	0x010e001b	16	Sample size for EDID while testing	

6.9.2 TSI_DP14_SRCCTS_TIMEOUT

```
TSI_DP14_SRCCTS_TIMEOUT                          0x010e0000
unsigned int dp_ll_cts_srcdut_timeout            U32
4 bytes                                          RW
```

Description

Defines the timeout for DP LL CTS Source DUT tests. The timeout is defined as milliseconds, and has a default setting of 5000ms. Please note that if the test contains internal iterations, this timeout is re-used for each iteration.

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.3 TSI_DP14_SRCCTS_MAX_LANES

```
TSI_DP14_SRCCTS_MAX_LANES                        0x010e0001
unsigned int dp_ll_cts_crcdut_max_lanes          U32
4 bytes                                          RW
```

Description

Defines the maximum number of lanes supported by the DUT. Typical settings are 1, 2 and 4. Default value is 4.

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.4 TSI_DP14_SRCCTS_MAX_LINK_RATE

```
TSI_DP14_SRCCTS_MAX_LINK_RATE                  0x010e0002
unsigned int dp_ll_cts_srcdut_max_link_rate    U32
4 bytes                                        RW
```

Description

Defines the maximum link rate as multiplier for 0.27Gbps. Typical settings are 6 (RBR), 10 (HBR), 20 (HBR2) and 30 (HBR3). Notice that HBR3 link rate is usable only with UCD-400. The default setting is 20 (HBR2).

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.5 TSI_DP14_SRCCTS_DUT_CAPS

```
TSI_DP14_SRCCTS_DUT_CAPS                       0x010e0003
unsigned int dp_ll_cts_srcdut_dut_caps         U32
4 bytes                                        RW
```

Description

Defines the DUT capabilities as flags. Default setting is zero. Please see table below for defined capability flags:

Bits	Description
0	Voltage swing level 3 support. 0 = Not supported, 1 = Supported.
1	Pre-emphasis level 3 support. 0 = Not supported, 1 = Supported.
2	Fixed timing DUT. 0 = DUT is not fixed timing, 1 = DUT is fixed timing.
3	Spread Spectrum support. 0 = Not supported, 1 = Supported.
4	Indicates support for video format change without link training. 0 = Not supported, 1 = Supported.
5	Indicates support for lane count reduction without link training. 0 = Not supported, 1 = Supported.
6	E-DDC protocol is supported. 0 = Not supported, 1 = Supported.
7	DUT Supports Audio Info Frame for 2 channel audio transmission. 0 = Not supported, 1 = Supported.
8	DUT device type. 0 = Normal, 1 = Type-C device.
31:9	RESERVED, SET TO ZERO

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.6 TSI_DP14_SRCCTS_DUT_TA

```
TSI_DP14_SRCCTS_DUT_TA                          0x010e0004
unsigned int dp_ll_cts_srcdut_dut_ta            U32
4 bytes                                         RW
```

Description

Defines the DUT Test automation capabilities as flags. Default setting is zero. Please see table below for defined test automation capabilityu flags:

Bits	Description
0	Test automation flag “TEST_LINK_TRAINING”. 1 = Supported, 0 = Not supported.
1	Test automation flag “TEST_EDID_READ”. 1 = Supported, 0 = Not supported.
2	Test automation flag “TEST_VIDEO_PATTERN”. 1 = Supported, 0 = Not supported.
4:3	Event signalling DUT ready: 0 = NONE, 1 = EDID, 2 = Link Training end, 3 = Active video.
5	Test automation flag “TEST_AUDIO_PATTERN”. 1 = Supported, 0 = Not supported.
31:5	RESERVED, SET TO ZERO

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.7 TSI_DP14_SRCCTS_LONG_HPD_PULSE

```
TSI_DP14_SRCCTS_LONG_HPD_PULSE                  0x010e0005
unsigned int dp_ll_cts_srcdut_long_hpd          U32
4 bytes                                         RW
```

Description

Defines the duration of long HPD pulses generated by the tests. The duration is defined in millieconds. Default setting is 1000ms.

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.8 TSI_DP14_SRCCTS_LT_START_TIMEOUT

```
TSI_DP14_SRCCTS_LT_START_TIMEOUT                0x010e0006
unsigned int dp_ll_cts_srcdut_lt_strt_timeout   U32
4 bytes                                         RW
```

Description

Defines a timeout for Link Training start, in milliseconds. Default setting is 5000ms.

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.9 TSI_DP14_SRCCTS_TEST_CYCLE_DELAY

```
TSI_DP14_SRCCTS_TEST_CYCLE_DELAY              0x010e0007
unsigned int dp_ll_cts_srcdut_cycle_delay     U32
4 bytes                                       RW
```

Description

Defines an internal cycle delay used within tests that repeat a number of test steps. The delay is defined in milliseconds. Default setting is 3000ms.

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.10 TSI_DP14_SRCCTS_COLOR_FORMATS

```
TSI_DP14_SRCCTS_COLOR_FORMATS                 0x010e0008
unsigned int dp_ll_cts_srcdut_color_formats   U32
4 bytes                                       RW
```

Description

Defines number of color formats programmed into the CI's TSI_DP14_SRCCTS_COLOR_MODE_0 to TSI_DP14_SRCCTS_COLOR_MODE_31. The default setting is 2.

Important: *The CTS Specification defines two mandatory color formats: 0x0 and 0x20. These color-format definitions must be present in the color mode table.*

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.11 TSI_DP14_SRCCTS_FAILSAFE_MODE

```
TSI_DP14_SRCCTS_FAILSAFE_MODE                 0x010e0009
unsigned int dp_ll_cts_srcdut_failsafe_mode   U32
4 bytes                                       RW
```

Description

Defines the fail-safe mode ID. The default setting is 1 (= 640 x 480 @ 60 Hz, 18 BPP). Please see 6.9.30 LL CTS Test resolution ID values for the mode definitions.

Important: *This setting should not be modified.*

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.12 TSI_DP14_SRCCTS_MAX_RESOLUTION

```
TSI_DP14_SRCCTS_MAX_RESOLUTION                0x010e000a
unsigned int dp_ll_cts_srcdut_max_resolution  U32
4 bytes                                       RW
```

Description

Defines the maximum resolution supported by the DUT. Default setting is 5 (= 1920 x 1080 @ 60Hz, 24 BPP). Please see 6.9.30 LL CTS Test resolution ID values for the mode definitions.

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.13 TSI_DP14_SRCCTS_*L_MOST_PACKED

```
TSI_DP14_SRCCTS_1L_MOST_PACKED                0x010e0020
TSI_DP14_SRCCTS_2L_MOST_PACKED                0x010e0021
TSI_DP14_SRCCTS_4L_MOST_PACKED                0x010e0022
unsigned int dp_ll_cts_src_dut_most_packed    U32
4 bytes                                       RW
```

Description

These three CI's define the most packed timings for 1, 2 and 4 lane modes. Default for 1 lane is 8 (= 1280 x 800 @ 60Hz, 18BPP). Default 2 lanes is 12 (= 1280 x 1024 @ 60Hz, 24BPP). Default for 4 lanes is 18 (= 2048 x 1536 @ 60Hz, 24BPP). Please see 6.9.30 LL CTS Test resolution ID values for the mode definitions.

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.14 TSI_DP14_SRCCTS_*L_*BR*

```
TSI_DP14_SRCCTS_1L_RBR                          0x010e0023
TSI_DP14_SRCCTS_2L_RBR                          0x010e0024
TSI_DP14_SRCCTS_4L_RBR                          0x010e0025
TSI_DP14_SRCCTS_1L_HBR                          0x010e0026
TSI_DP14_SRCCTS_2L_HBR                          0x010e0027
TSI_DP14_SRCCTS_4L_HBR                          0x010e0028
TSI_DP14_SRCCTS_1L_HBR2                         0x010e0029
TSI_DP14_SRCCTS_2L_HBR2                         0x010e002a
TSI_DP14_SRCCTS_4L_HBR2                         0x010e002b
TSI_DP14_SRCCTS_1L_HBR3                         0x010e002c
TSI_DP14_SRCCTS_2L_HBR3                         0x010e002d
TSI_DP14_SRCCTS_4L_HBR3                         0x010e002e
unsigned int dp_ll_cts_src_dut_timestamp_gen  U32
4 bytes                                         RW
```

Description

These 12 CI's define the video modes used for timestamp generations for RBR, HBR, HBR2 and HBR3 link rates with 1, 2 and 4 lanes configurations. Please see 6.9.1 Test ID definitions, configuration items for defaults. Please see 6.9.30 LL CTS Test resolution ID values for the mode definitions.

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.15 TSI_DP14_SRCCTS_COLOR_MODE_*

```
TSI_DP14_SRCCTS_COLOR_MODE_0                    0x010e0038
TSI_DP14_SRCCTS_COLOR_MODE_1                    0x010e0039
TSI_DP14_SRCCTS_COLOR_MODE_2                    0x010e003a
TSI_DP14_SRCCTS_COLOR_MODE_3                    0x010e003b
TSI_DP14_SRCCTS_COLOR_MODE_4                    0x010e003c
TSI_DP14_SRCCTS_COLOR_MODE_5                    0x010e003d
TSI_DP14_SRCCTS_COLOR_MODE_6                    0x010e003e
TSI_DP14_SRCCTS_COLOR_MODE_7                    0x010e003f
TSI_DP14_SRCCTS_COLOR_MODE_8                    0x010e0040
TSI_DP14_SRCCTS_COLOR_MODE_9                    0x010e0041
TSI_DP14_SRCCTS_COLOR_MODE_10                   0x010e0042
TSI_DP14_SRCCTS_COLOR_MODE_11                   0x010e0043
TSI_DP14_SRCCTS_COLOR_MODE_12                   0x010e0044
TSI_DP14_SRCCTS_COLOR_MODE_13                   0x010e0045
TSI_DP14_SRCCTS_COLOR_MODE_14                   0x010e0046
TSI_DP14_SRCCTS_COLOR_MODE_15                   0x010e0047
TSI_DP14_SRCCTS_COLOR_MODE_16                   0x010e0048
TSI_DP14_SRCCTS_COLOR_MODE_17                   0x010e0049
TSI_DP14_SRCCTS_COLOR_MODE_18                   0x010e004a
TSI_DP14_SRCCTS_COLOR_MODE_19                   0x010e004b
TSI_DP14_SRCCTS_COLOR_MODE_20                   0x010e004c
TSI_DP14_SRCCTS_COLOR_MODE_21                   0x010e004d
TSI_DP14_SRCCTS_COLOR_MODE_22                   0x010e004e
TSI_DP14_SRCCTS_COLOR_MODE_23                   0x010e004f
TSI_DP14_SRCCTS_COLOR_MODE_24                   0x010e0050
TSI_DP14_SRCCTS_COLOR_MODE_25                   0x010e0051
TSI_DP14_SRCCTS_COLOR_MODE_26                   0x010e0052
TSI_DP14_SRCCTS_COLOR_MODE_27                   0x010e0053
TSI_DP14_SRCCTS_COLOR_MODE_28                   0x010e0054
TSI_DP14_SRCCTS_COLOR_MODE_29                   0x010e0055
TSI_DP14_SRCCTS_COLOR_MODE_30                   0x010e0056
TSI_DP14_SRCCTS_COLOR_MODE_31                   0x010e0057
unsigned int dp_ll_cts_src_dut_color_defs       U32
4 bytes                                         RW
```

Description

Defines the color modes used with CTS tests. Colors modes 0 and 1 must be set to 0 and 0x20, which defined the mandatory color modes; These are set by default. All other configuration items have default value of zero. Please see 6.9.29 Color mode bit-field definitions for details on how to encode the color information.

Important: *This CI is still potential subject for changes due to the CTS specification not being officially ready.*

6.9.16 TSI_DP14_SRCCTS_MIN_SAMPLE_RATE

```
TSI_DP14_SRCCTS_MIN_SAMPLE_RATE                0x010e0010
unsigned int dp_ll_min_srate                   U32
4 bytes                                        RW
```

Description

Defines the minimum sample rate supported by the DUT in Hz. Allowed values are 32000, 44100, 48000, 88200, 96000 or 192000 Hz. Default setting is 44100 Hz.

6.9.17 TSI_DP14_SRCCTS_MAX_SAMPLE_RATE

```
TSI_DP14_SRCCTS_MAX_SAMPLE_RATE                0x010e0011
unsigned int dp_ll_max_srate                   U32
4 bytes                                        RW
```

Description

Defines the maximum sample rate supported by the DUT in Hz. Allowed values are 32000, 44100, 48000, 88200, 96000 or 192000 Hz. Default setting is 44100 Hz.

6.9.18 TSI_DP14_SRCCTS_CHANNELS1

```
TSI_DP14_SRCCTS_CHANNELS1                      0x010e0012
unsigned int dp_ll_chn1                        U32
4 bytes                                        RW
```

Description

Defines the minimum number of channels for minimum sample rate. Default is 2 channels (Stereo). Highest valid channel count is 8. If the number is different from 2 channels, then the appropriate channel allocation must be provided through the CI 6.9.19 TSI_DP14_SRCCTS_CH_ALLOC1.

6.9.19 TSI_DP14_SRCCTS_CH_ALLOC1

```
TSI_DP14_SRCCTS_CH_ALLOC1                      0x010e0013
unsigned int dp_ll_alloc1                      U32
4 bytes                                        RW
```

Description

Defines the channel allocation for minimum number of channels and sample rate. Default setting is 0. Please refer to 6.9.28 Channel allocation bits for details on how to assign the allocation bits.

6.9.20 TSI_DP14_SRCCTS_CHANNELS2

TSI_DP14_SRCCTS_CHANNELS2	0x010e0014
unsigned int dp_ll_chn2	U32
4 bytes	RW

Description

Defines the maximum number of channels for minimum sample rate. Default is 2 channels (Stereo). Highest valid channel count is 8. If the number is different from 2 channels, then the appropriate channel allocation must be provided through the CI 6.9.21 TSI_DP14_SRCCTS_CH_ALLOC2.

6.9.21 TSI_DP14_SRCCTS_CH_ALLOC2

TSI_DP14_SRCCTS_CH_ALLOC2	0x010e0015
unsigned int dp_ll_alloc2	U32
4 bytes	RW

Description

Defines the channel allocation for maximum number of channels with minimum sample rate. Default setting is 0. Please refer to 6.9.28 Channel allocation bits for details on how to assign the allocation bits.

6.9.22 TSI_DP14_SRCCTS_CHANNELS3

TSI_DP14_SRCCTS_CHANNELS3	0x010e0016
unsigned int dp_ll_chn3	U32
4 bytes	RW

Description

Defines the minimum number of channels for maximum sample rate. Default is 2 channels (Stereo). Highest valid channel count is 8. If the number is different from 2 channels, then the appropriate channel allocation must be provided through the CI 6.9.23 TSI_DP14_SRCCTS_CH_ALLOC3.

6.9.23 TSI_DP14_SRCCTS_CH_ALLOC3

TSI_DP14_SRCCTS_CH_ALLOC3	0x010e0017
unsigned int dp_ll_alloc3	U32
4 bytes	RW

Description

Defines the channel allocation for minimum number of channels with maximum sample rate. Default setting is 0. Please refer to 6.9.28 Channel allocation bits for details on how to assign the allocation bits.

6.9.24 TSI_DP14_SRCCTS_CHANNELS4

```
TSI_DP14_SRCCTS_CHANNELS4                    0x010e0018
unsigned int dp_ll_chn4                      U32
4 bytes                                      RW
```

Description

Defines the maximum number of channels for maximum sample rate. Default is 2 channels (Stereo). Highest valid channel count is 8. If the number is different from 2 channels, then the appropriate channel allocation must be provided through the CI 6.9.25 TSI_DP14_SRCCTS_CH_ALLOC4.

6.9.25 TSI_DP14_SRCCTS_CH_ALLOC4

```
TSI_DP14_SRCCTS_CH_ALLOC4                    0x010e0019
unsigned int dp_ll_alloc4                    U32
4 bytes                                      RW
```

Description

Defines the channel allocations for maximum number of channels and sample rate. Default setting is 0. Please refer to 6.9.28 Channel allocation bits for details on how to assign the allocation bits.

6.9.26 TSI_DP14_SRCCTS_AUDIO_TEST_PATTERN

```
TSI_DP14_SRCCTS_AUDIO_TEST_PATTERN           0x010e001a
unsigned int dp_ll_audio_test_pattern        U32
4 bytes                                      RW
```

Description

Defines the audio test pattern. Default setting is zero. Please see table below for allowed settings:

Value	Description
0	Operator specific waveform
1	Sawtooth waveform

6.9.27 TSI_DP14_SRCCTS_EDID_SAMPLE_SIZE

```
TSI_DP14_SRCCTS_EDID_SAMPLE_SIZE                0x010e001b
unsigned int dp_ll_edid_sample_size             U32
4 bytes                                         RW
```

Description

Defines sample size used for EDID configuration while testing. The size is indicated as number of bits per sample. Default setting is 16. Valid settings are 16, 20 or 24.

6.9.28 Channel allocation bits

The audio parameters contain four pairs of audio channel count and channel allocation values. Please notice that maximum channel count allowed is 8. For two channels, the channel allocation of zero is considered valid; In this case, stereo is assumed (FL/FR).

Allocation value	Channel count/bit	Channel assignments
0x001	2	FL/FR. (Front Left, Front Right)
0x002	1	LFE. (Low Frequency Effects)
0x004	1	FC. (Front Center)
0x008	2	RL/RR. (Rear Left, Rear Right).
0x010	1	RC. (Rear Center).
0x020	2	FLC/FRC. (Front Left off Center, Front Right off Center).
0x040	2	RLC/RRC (Rear Left off Center, Rear Right off Center).
0x080	2	FLW/FRW (Front Left Wide, Front Right Wide)
0x100	2	FLH/FRH (Front Left High, Front Right High)
0x200	1	TC (Top Center)
0x400	1	FCH (Front Center High)

```
Example: Channel allocation bits and channel count for 5.1 surround audio
system.

In a system like this, the speaker positions are Front left, Front right,
Front Center, Rear Left, Read Right and Low Frequency Effects.

For this system we need to select FL/FR + LFE + FC + RL/RR lines from the
above table. The values from the table are then added together to form the
channel allocation bits and the channel count as follows:

Channel_Alloc = (Allocation_bit(FL/FR)) | (Allocation_bit(LFE)) |
                (Allocation_bit(FC)) | (Allocation_bit(RL/RR));

Channel_Count = (Channel_count(FL/FR)) + (Channel_count(LFE)) +
                (Channel_count(FC)) + (Channel_count(RL/RR));
```

6.9.29 Color mode bit-field definitions

The color format bits are define as follows:

Bits	Description	
0	RESERVED – Set to zero	
2:1	Color space.	
	0	RGB
	1	YCbCr 4:2:2
	2	YCbCr 4:4:4
	3	RESERVED
3	Dynamic range	
	0	VESA
	1	CEA
4	YCbCr coefficients	
	0	ITU-601
	1	ITU-709
7:5	Colordepth	
	0	18 bits per pixel
	1	24 bits per pixel
	2	30 bits per pixel
	3	36 bits per pixel
	4	48 bits per pixel
	*	RESERVED, Do not use.
31:8	RESERVED – Set to zero	

6.9.30 LL CTS Test resolution ID values

The supported resolution ID values are listed in the table below:

ID	H-Res	V-Res	Frequency, Hz	Color depth, BPP	Comment
0	-	-	-	-	No timing – Use to fill un-used entries.
1	640	480	60	18	VGA; Used as fail safe timing
2	848	480	60	24	-
3	1280	720	60	24	-
4	1280	960	60	24	-
5	1920	1080	60	24	-
6	1920	1440	60	24	-
7	1920	1080	120	24	-
8	1280	800	60	18	-
9	1280	768	60	18	-
10	800	600	60	30	-
11	1024	768	60	30	-
12	1280	1024	60	24	-
13	1360	768	60	30	-
14	1280	800	60	30	-
15	1400	1050	60	24	-
16	1280	768	60	30	-
17	1600	1200	60	18	-
18	2048	1536	60	24	-
19	1792	1344	60	24	-
20	1600	1200	60	30	-
21	3840	2160	30	24	-
22	3840	2160	60	24	-

7 TSI PROGRAMMING

This section defines mechanisms used in TSI that might not be otherwise clear from the other parts of the reference manual. The intent is to clarify the operation of the more complex parts by providing additional descriptions and details on intended uses.

7.1 Operator feedback during test execution

Some of the new TSI tests may require operator feedback and/or operator actions during test execution. This section describes the implementation details in TSI that are used to carry out the operator communications. The provided solution only prompts operator to carry out operations on the DUT side, but the intention of this mechanism is to enable test system manufacturers to fully automate test execution even when the test requires an operation to be performed on the DUT side.

7.1.1 Operator feedback implementation in TSI

TSI is planned to support more than one way of operator intervention / feedback mechanisms. The CI named TSI_TS_OFMODE is used to select which method is used. Currently three methods are supported: TSI_OFMODE_RUN_EXTERNAL (default), TSI_OFMODE_RUN_INTERNAL and TSI_OFMODE_RUN_CALL_PROCEEDURE, having values of 1, 2, and 3. In TSI_OFMODE_RUN_EXTERNAL mode, TSI will run an external application. TSI_OFMODE_RUN_INTERNAL mode is used exclusively with TSI-X automation in loopback mode (same device output to same device input) for devices such as the UDC-400: The test and feedback scripts are run within the same process. TSI_OFMODE_RUN_CALL_PROCEEDURE can be used by applications using the TSI.dll to have the tests call certain functions when feedback is required: The function signature is int(__stdcall *pfn)(char *str). A default command line application is distributed with TSI, along with its source code. TSI itself can be the external application (see 7.3.6 Running Tests).

7.1.2 Selecting which application TSI will run

By default, TSI will use its own operator feedback application or function for all operator feedback requests. However, it is possible to define separate applications for each operator feedback request type. The definition is controlled by three separate CI's. The first of these is "TSI_TS_OFMODE" as described above. The second is "TSI_TS_OF_REQ_ID", which defines the request ID related to the application setting. The default setting is -1, which means that the application setting applies to all request ID's. The third CI is "TSI_TS_OF_EXT_APP", which contains the path and name of the executable file to be started as a NULL terminated string. These definitions are in the device scope, so each test device will have its own set of application definitions. (The same applies to setting callback function pointers with TSI_OFMODE_RUN_CALL_PROCEEDURE.)

In short, to use a single application for all operator feedback requests:

1. Write 1 to "TSI_TS_OFMODE" CI (default).
2. Write -1 to "TSI_TS_OF_REQ_ID" CI.
3. Write a null terminated string containing full path and name of your application to "TSI_TS_OF_EXT_APP" CI.

To configure an application for any operator feedback request (this step can be repeated to create multiple settings) :

1. Write 1 to "TSI_TS_OFMODE" CI (default).
2. Write the request ID to "TSI_TS_OF_REQ_ID".
3. Write a null terminated string containing full path and name of your application to "TSI_TS_OF_EXT_APP" CI.

7.1.3 External application requirements

When TSI calls an external application, it passes information about the request as a command-line parameter containing a number of key-value pairs.

The external application is expected to parse the key-value pairs and result in the DUT carrying out the requested operation. It is up to the designer of this application to decide how to implement it.

The external application is expected to return one of the response values listed in the requested information based on the outcome of the operation.

TSI Default application operation

The default application will show a command-prompt window with a message to the test operator, and wait for the test operator to enter a response. Possible responses are:

- *'Enter'* only → "Proceed" response.
- *'P'+ 'Enter'* → "Pass" response. (Note: Case-insensitive)
- *'F'+ 'Enter'* → "Fail" response. (Note: Case-insensitive)

Entering an invalid response, or a response that is not listed in the request information will not be accepted and the application will repeat the prompt for a valid response.

7.1.4 Request parameters

When a test requires operator intervention to configure the DUT and/or other operator feedback, TSI will run the defined application with command-line parameters that define the contents of the request. The command-line parameters are passed as a key-value pair script. The key-value pair script assumes the following rules in its syntax:

- "KEY" and "VALUE" -fields are separated with the equal character ('=').
- "VALUE" -field is followed by a semi-colon (';').
- "KEY" and "VALUE" -field pair must be on a single line of text.
- If either the "KEY" or "VALUE" -field contains spaces, the field must be enclosed with single quote-marks.
- Strings: The closing single-quote-mark also marks end of the field being processed.
- White-spaces characters are allowed anywhere outside "KEY" and "VALUE" -fields.
- New-lines (and white-spaces) may always appear before beginning of the next "KEY" -field. For example, at start of file and after semi-colon (';') ending the previous key-value pair.

The "KEY" -fields are always character strings, and "VALUE" -fields are decimal numeric values. Please see the table on the next page for a list of the "KEY"-fields generated by TSI, along with the meaning of the associated "VALUE"-fields.

(Continued...)

(...Continued)

Key	Value / Description		Reference
"op"	The value is a request ID that identifies what is being requested for the operator to do and/or evaluate. Currently valid ID's are listed below.		-
	1	Request for DUT to start link training with given lane count and link speed	7.1.5.1
	2	Request for DUT to transmit test pattern #1 in given video mode	7.1.5.2
	3	Request for DUT to read EDID from TE	7.1.5.3
	4	Request for DUT to reduce number of lanes being used	7.1.5.4
	5	Request for DUT to increase number of lanes being used	7.1.5.5
	6	Request for DUT to transmit test pattern #1 in given video and color mode	7.1.5.6
	7	Request for DUT to enter power save mode	7.1.5.7
	8	Request for DUT to exit power save mode	7.1.5.8
	9	Request for DUT to transmit test pattern #1 in given video mode and color depth	7.1.5.9
	10	Request for DUT to transmit video and audio with defined parameters.	7.1.5.10
	11	Request for operator to check audio test pattern playback and respond with "pass" if audio playback is correct, and with "fail" if audio playback is not correct.	7.1.5.11
	12	Request for operator to check video pattern and respond with "pass" if video playback is correct, and with "fail" if video playback is not correct.	7.1.5.12
	13	Request for operator to play the "ping test pattern" audio stream three times and verify it played correctly. If the playback is correct, respond with "pass" or if the playback was not correct, respond with "fail"	7.1.5.13
	14	Request for DUT to write FEC_READY bit in DPCD register 0x120	7.1.5.14
	15	Request for DUT to send ENABLE_FEC sequence	7.1.5.15
	16	Request for DUT to send DISABLE_FEC sequence	7.1.5.16
	17	Request for DUT to enable the DSC Regime	7.1.5.17
	18	Request for DUT to transmit DSC compressed image with the specified video mode using the specified color mode.	7.1.5.18
"res_x"	The value indicates the width of the active area of the requested video mode in terms of number of pixels		-
"res_y"	The value indicates the height of the active area of the requested video mode in terms of number of pixels		-
"res_bpp"	The value indicates the number of bits per pixel for the requested video mode		-
"res_frate"	The value indicates the frame-rate of the requested video mode as milli-Hertz		-
"col_format"	Indicates requested color format as an ID value. Currently valid ID's are listed below:		-
	0	RGB	-
	1	YCbCr 4:2:2	-
	2	YCbCr 4:4:4	-
"col_range"	Indicates the dynamic range standard for the requested color space as an ID value. Currently valid ID's are listed below:		-
	0	VESA	-
	1	CEA	-
"col_yc"	Indicates the colorimetry standard for the requested color space as an ID value. Currently valid ID's are listed below:		-
	0	ITU-601	-
	1	ITU-709	-

(Continued…)

(...Continued)

Key	Value / Description		Reference
“col_bpp”	Indicates color-depth as bits per pixel for the requested color space		-
“dp_lanes”	Indicates number of DP lanes to be used in the next link training performed by the DUT		-
“dp_linkrate”	Indicates link-rate of DP lanes to be used in the next link training performed by the DUT. The link rate is provided as a multiplier of 0.27Gbps.		-
“aud_pat”	Indicates the requested audio pattern type		-
	0	Operator specific	
	1	Sawtooth pattern	
“aud_chn”	Indicates number of audio channels to send. Typical range from 2 to 8.		-
“aud_freq”	Indicates the sampling rate of the audio stream to send, as Hz.		-
“aud_bits”	Indicates the sample size as number of bits.		-
“aud_alloc”	Channel allocation bits to be used. The bits are passed as a decimal number. ***Important:*** *A value of zero mean no channel allocations bits to be used – This is valid for stereo audio.*		-
	Bit	Purpose	
	0	FL + FR (Front-left and Front-right channels present)	
	1	LFE (Low-frequency-effects channel present)	
	2	FC (Front-center channel present)	
	3	RL + RR (Read-left and Rear-right channels present)	
	4	RC (Rear-center channel present)	
	5	FLC + FRC (Front-left-of-center and Front-right-of-center channels present)	
	6	RLC + RRC (Rear-left-of-center and Rear-right-of-center channels present)	
	7	FLW + FRW (Front-left-wide and Front-right-wide channels present)	
	8	FLH + FRH (Front-left-high and Front-right-high channels present)	
	9	TC (Top-center channel present)	
	10	FHC (Front-high-center channel present)	
“tim_std”	Indicates additional details about video timing to select. Typically, this is used with tests related to DSC.		-
	0	CTA. See CTA-861-G for additional details	
	1	Use VESA / CVT RB1 Coordinated video timing	
	2	Use VESA / CVT RB2 Coordinated video timing	
“aud_ppr_1” … “aud_ppr_8”	Indicates the pattern period as number of samples. ***Important:*** *The values are only present for number of channels requested. Do not expect there to always be same number of period keys.* ***Important:*** *the values will not be passed if the requested audio pattern is not “sawtooth”.*		-
“exit_proceed”	Defines expected exit code for “proceed” response		-
“exit_pass”	Defines expected exit code for “pass” response.		-
“exti_fail”	Defiens expected exit code for “fail” response.		-

7.1.5 Request parameter details

7.1.5.1 Request ID 1

Request for DUT to start link training with given lane count and link speed. Passed parameters include at least the following:

- "op" – ID for this request (1)
- "dp_lanes" – Number of lanes to use. Typically 1, 2 or 4.
- "dp_linkrate" – Link rate multiplier for 0.27 Gbps
- "exit_proceed" – Value for the application to return as its exit code

7.1.5.2 Request ID 2

Request for DUT to transmit test pattern #1 in given video mode. Passed parameters include at least the following:

- "op" – ID for this request (2)
- "res_x" – Number of horizontal pixels on the active area
- "res_y" – Number of vertical pixels on the active area
- "res_frate" – Number of frames per second, as milli-hertz (60 fps = 60000)
- "exit_proceed" – Value for the application to return as its exit code

7.1.5.3 Request ID 3

Request for DUT to read EDID from TE. Passed parameters include at least the following:

- "op" – ID for this request (3)
- "exit_proceed" – Value for the application to return as its exit code

7.1.5.4 Request ID 4

Request for DUT to reduce number of lanes being used. Passed parameters include at least the following:

- "op" – ID for this request (4)
- "exit_proceed" – Value for the application to return as its exit code

7.1.5.5 Request ID 5

Request for DUT to increase number of lanes being used. Passed parameters include at least the following:

- "op" – ID for this request (5)
- "exit_proceed" – Value for the application to return as its exit code

7.1.5.6 Request ID 6

Request for DUT to transmit test pattern #1 in given video and color mode. Passed parameters include at least the following:

- "op" – ID for this request (6)
- "res_x" – Number of horizontal pixels on the active area
- "res_y" – Number of vertical pixels on the active area
- "res_frate" – Number of frames per second, as milli-hertz (60 fps = 60000)
- "res_bpp" – Number of bits per pixel, as number of bits
- "col_format" – Indicates the wanted pixel color formatting
- "col_range" – Indicates dynamic range (VESA/CEA)
- "col_yc" – Indicates YCbCr coefficients (ITU-601/ITU-709)
- "exit_proceed" – Value for the application to return as its exit code

7.1.5.7 Request ID 7

Request for DUT to enter power save mode. Passed parameters include at least the following:

- "op" – ID for this request (7)
- "exit_proceed" – Value for the application to return as its exit code

7.1.5.8 Request ID 8

Request for DUT to exit power save mode. Passed parameters include at least the following:

- "op" – ID for this request (8)
- "exit_proceed" – Value for the application to return as its exit code

7.1.5.9 Request ID 9

Request for DUT to transmit test pattern #1 in given video mode and color depth. Passed parameters include at least the following:

- "op" – ID for this request (9).
- "res_x" – Number of horizontal pixels on the active area.
- "res_y" – Number of vertical pixels on the active area.
- "res_frate" – Number of frames per second, as milli-hertz (60 fps = 60000).
- "res_bpp" – Number of bits per pixel, as number of bits.
- "exit_proceed" – Value for the application to return as its exit code.

7.1.5.10 Request ID 10

Request for DUT to transmit video and audio with defined parameters. The following parameters are included:

- "op" – ID for this request (10).
- "res_x" – Number of horizontal pixels on the active area.
- "res_y" – Number of vertical pixels on the active area.
- "res_frate" – Number of frames per second, as milli-hertz (60 fps = 60000).
- "res_bpp" – Number of bits per pixel, as number of bits.
- "aud_pat" – Indicates the audio pattern to be used.
- "aud_chn" – Indicates the number of audio channels to be used.
- "aud_freq" – Indicates the sampling rate to be used.
- "aud_bits" – Indicates the sample size, in bits, to be used.
- "aud_alloc" – Indicates the channel allocation bits to be used.
- "aud_ppr_1" … "aud_ppr_8" – Indicates the pattern periods as count of samples for each channel to be used. NOTE: These values are present only when the requested audio pattern is "sawtooth".
- "exit_proceed" – Value for the application to return as its exit code.

7.1.5.11 Reqeust ID 11

Request for operator to check audio test pattern playback and respond with "pass" if audio playback is correct, and with "fail" if audio playback is not correct. The following parameters are included:

- "op" – ID for this request (11)
- "exit_pass" – Value for the application to return for "pass" response.
- "exit_fail" – Value for the application to return for "fail" response.

7.1.5.12 Request ID 12

Request for operator to check video pattern and respond with “pass” if video playback is correct, and with “fail” if video playback is not correct. The following parameters are included:

- “op” – ID for this request (12)
- “exit_pass” – Value for the application to return for “pass” response.
- “exit_fail” – Value for the application to return for “fail” response.

7.1.5.13 Request ID 13

Request for operator to play the “ping test pattern” audio stream three times and verify it played correctly. If the playback is correct, respond with “pass” or if the playback was not correct, respond with “fail”. The following parameters are included:

- “op” – ID for this request (13)
- “exit_pass” – Value for the application to return for “pass” response.
- “exit_fail” – Value for the application to return for “fail” response.

7.1.5.14 Request ID 14

Request for DUT to write FEC_READY bit in DPCD register 0x120 FEC_CONFIGURATION. The following parameters are included:

- “op” – ID for this request (14)
- “exit_proceed” – Value for the application to return as its exit code.

7.1.5.15 Request ID 15

Request for DUT to send the ENABLE_FEC sequence as described int the DP specifications. The following parameters are included:

- “op” – ID for this request (15)
- “exit_proceed” – Value for the application to return as its exit code.

7.1.5.16 Request ID 16

Request for DUT to send the DISABLE_FEC sequence as described in the DP specficiations. The following parameters are included:

- “op” – ID for this request (16)
- “exit_proceed” – Value for the application to return as its exit code.

7.1.5.17 Request ID 17

Request for DUT to enable DSC regime.

- “op” – ID for this request (17)
- “exit_proceed” – Value for the application to return as its exit code.

7.1.5.18 Request ID 18

Request for DUT to transmit DSC compressed image with the specified video mode using the specified color mode.

- "op" – ID for this request (18)
- "res_x" – Number of horizontal pixels on the active area.
- "res_y" – Number of vertical pixels on the active area.
- "res_frate" – Number of frames per second, as milli-hertz (60 fps = 60000).
- "res_bpp" – Number of bits per pixel, as number of bits.
- "col_format" – Indicates the wanted pixel color formatting
- "col_range" – Indicates dynamic range (VESA/CEA)
- "col_yc" – Indicates YCbCr coefficients (ITU-601/ITU-709)
- "tim_std" – Indicates which kind of timing standard should be selected.
- "exit_proceed" – Value for the application to return as exit code.

7.1.6 Operator Feedback configuration items

The following configuration items are used to configure the operator feedback mechanism.

7.1.6.1 TSI_TS_OF_MODE

```
TSI_TS_OF_MODE                                  0x215
unsigned int of_mode                            U32
4 bytes                                         RW
```

Description

This value defines how the Operator Feedback mechanism invokes external utilities and/or scripts. Currently supported methods are:

ID	Method name / Define	Description
1	TSI_OFMODE_RUN_EXTERNAL	(Default) TSI will run an external program if a test requires operator feedback of intervention. Be default, the application "OPF_CMD.exe" is started.
2	TSI_OFMODE_RUN_INTERNAL	This mode is to be used only when both source and sink are Unigraf TE devices that are supported by TSI – For example, when testing repeater or branch DUT's. In this mode, the test and feedback automation scripts are run within the same process, avoiding a lot of overhead of starting new instance of TSI. By default, each operator feedback ID will get separate script file definition formed as "OperatorFeedBackRequest_ID_<X>", where X is replaced with the request's ID number.
3	TSI_OFMODE_CALLBACK	Callback function. This mode is used when a client software directly interacts with TSI.DLL. Please see 7.1.6.4 TSI_TS_OF_CALLBACK for additional information. By default, callback pointers are initialized to NULL.

Important: *When this CI is written, the default settings will be re-applied meaning that any changes made to external application names, script file names and callback pointers will be lost.*

7.1.6.2 TSI_TS_OF_REQ_ID

```
TSI_TS_OF_REQ_ID                                    0x216
int of_request_id                                   S32
4 bytes                                             RW
```

Description

Selects which request's configuration is accessed through TSI_TS_OF_EXT_APP. Please see table below for request ID values currently defined:

Request ID	Description	Reference
-1	Select All requests. This ID has special meaning for configuring the external applications, external scripts or Callback pointers. Generally, setting external application, external script or callback pointer while this CI is set to -1 will clear any previous settings and sets the default external application, external script or callback pointer. Please see 7.1.6.3 TSI_TS_OF_EXT_APP and 7.1.6.4 TSI_TS_OF_CALLBACK for additional details!	7.1.6.3 7.1.6.4
1	Select request for DUT to start link training with given lane count and link speed	7.1.5.1
2	Select request for DUT to transmit test pattern #1 in given video mode	7.1.5.2
3	Select request for DUT to read EDID from TE	7.1.5.3
4	Select request for DUT to reduce number of lanes being used	7.1.5.4
5	Select request for DUT to increase number of lanes being used	7.1.5.5
6	Select request for DUT to transmit test pattern #1 in given video and color mode	7.1.5.6
7	Select request for DUT to enter power save mode	7.1.5.7
8	Select request for DUT to exit power save mode	7.1.5.8
9	Select request for DUT to transmit test pattern #1 in given video mode and color depth	7.1.5.9
10	Select request for DUT to transmit video and audio with defined parameters.	7.1.5.10
11	Select request for operator to check audio test pattern playback and respond with "pass" if audio playback is correct, and with "fail" if audio playback is not correct.	7.1.5.11
12	Select request for operator to check video pattern and respond with "pass" if video playback is correct, and with "fail" if video playback is not correct.	7.1.5.12
13	Select request for operator to play the "ping test pattern" audio stream three times and verify it played correctly. If the playback is correct, respond with "pass" or if the playback was not correct, respond with "fail"	7.1.5.13
14	Request for DUT to write FEC_READY bit in DPCD register 0x120	7.1.5.14
15	Request for DUT to send ENABLE_FEC sequence	7.1.5.15
16	Request for DUT to send DISABLE_FEC sequence	7.1.5.16
17	Request for DUT to enable DSC regime.	7.1.5.17
18	Request for DUT to transmit DSC compressed image with the specified video mode using the specified color mode.	7.1.5.18

7.1.6.3 TSI_TS_OF_EXT_APP

TSI_TS_OF_EXT_APP	0x217
char external_app[]	ARRAY_U8
260 bytes	RW

Description

This CI has slightly different operations depending on which setting is applied in 7.1.6.1 TSI_TS_OF_MODE.

OF_MODE Method 1:

Contains a NULL terminated string defining the path and name of the application to be executed for the request ID defined in TSI_TS_OF_REQ_ID configuration item.

Important: *When the selected request ID is -1 (TSI_TS_OF_REQ_ID), accessing this CI will have special processing applied to it as defined below*

- *On write, the internal list of external applications is cleared, and then the default external application is added.*
- *On read, this CI will return the default application path and name.*

OF_MODE Method 2:

Contains a NULL terminated string defining the name of the script file to be executed for the request ID defined in TSI_TS_OF_REQ_ID configuration item. Path may be optionally included as part of the script filename.

Important: *When the selected request ID is -1 (TSI_TS_OF_REQ_ID), accessing this CI will have special processing applied to it as defined below*

- *On write, the value being written is extended with each applicable operator feedback ID. For example, if writing "MyScript_", the resulting setting for feedback ID 1 is "MyScript_1", and for feedback ID 2 "MyScript_2", and so on for each supported feedback ID.*
- *On read, this CI will return the "printf(...) compatible" formatting string used to generate the script filenames.*

7.1.6.4 TSI_TS_OF_CALLBACK

```
TSI_TS_OF_CALLBACK                              0x218
int (__stdcall *Callback)(char *cmd)            PTR
Variable size                                   RW
```

Description

Contains a function pointer to be called by TSI defining the function to be called for request ID defined in TSI_TS_OF_REQ_ID configuration item. The default pointer for each request ID is NULL. If the pointer is NULL, no callback can be made and the test is automatically aborted.

The called function will receive pointer to a NULL terminated string of text. The string's content is the same as what is given to an external application through the command line. Please refer to 7.1.4 Request parameters for details about these values.

Imporant: *When the selected request ID -1 (TSI_TS_OF_REQ_ID), accessing this CI will have special processing applied to it as defined below.*

- *On write, the internal list of function pointers is cleared, and then the default function pointer is set.*
- *On read, this CI will return the default function pointer.*

Important: *TSI calls the function as an "unmanaged" function. If your application is "managed code", then it may be necessary to introduce additional marshaling or something other input/output processing depending on the used programming language. Please take extra care when facing this situation, as it is very easy to get wrong.*

Important: *TSI calls the function with "__stdcall" call type (In windows). It is important the called function is declared with compatible call type. If not, it is possible that the program stack gets corrupted, which in turn may lead to crash when calling to callback, or when returning to TSI from it.*

7.2 Extended scripting engine

Make sure the Unigraf Software Bundle is installed.

The TSI Extended Scripting Engine is an executable scripting language program, TSI.exe. Input is in the form of command line parameters. It is built on top of the Extended TSI API. On English versions of Windows, the 64 and 32 bit versions are typically found in the C:\Program Files (x86)\Unigraf\TSI\x64 and C:\Program Files (x86)\Unigraf\TSI\x86 directories.

In general, scripting is performed by opening a Command Prompt window and navigating to the directory containing the scripting examples.

A scripting environment is typically found at C:\Program Files (x86)\Unigraf\TSI\SDK\TSI_Automation_Script_Examples.

All input needs to be ASCII characters. Sometimes a text editor will add a BOM (Byte Order Mark) to the beginning of a file which contains non-ASCII characters. This will cause an error. In Notepad++ (a freely available text editing application) select the "encoding" tab to check and fix the BOM marker.

The "#" is used for comments. All input on a line after a "#" will be ignored. This means you must not try to print this symbol.

The TSI.exe return value (except for operator feedback runs covered later) is the number of tests failed. When running TE device built-in tests (ex. TSI -*c* .loadtd=filename.td), the global variable *failedtests* is incremented every time a test fails. General scripting errors that cause TSI.exeto terminate return a value of -1. If the program terminates because of a CI error (Configuration Item, covered later), the CI error return value is returned as a negative integer. Global variable *failedtesets* can be used to change the return value for successful runs (those with no scripting or CI errors):

```
-var failedtests=2
```

will set the pending return value to 2.

7.3 Getting started

To run TSI.*exe*, two *optional* text format files are generally used: *init.tsi* and *macros.tsi* . These need to be in the same directory from which TSI.*exe* is run. This is called the working directory. The location of inti.tsi can be changed (covered later). Note that TSI.exe does not need to be in the working directory if the Unigraf Software Bundle has installed.

Create a folder C:\Users\Karih\Documents\TestTSI (or any name you choose) in your documents directory (or anywhere else). Copy init.tsi and macros.tsi from C:\Program Files (x86)\Unigraf\TSI\SDK\TSI_Automation_Script_Examples\Configuration to directory TestTSI. Open a terminal and navigate to C:\Users\Karih\Documents\TestTSI. Edit init.tsi. To get information about the open TE devices available, on the command line type:

```
TSI -l
```

The output will contain (among other things) something like:

```
Dev0 = 1533c046 sink dp
Dev1 = 1533c046 source dp MST;
Dev2 = 1533c046 source dp SST;
Dev3 = 1938c748 sink dp;
Dev4 = 1938c748 sink hdmi;
```

```
Dev5 = 1938c748 sink hdmi_spdif;
```

Copy the lines you want to use to file init.tsi and name your devices. For example

```
SourceDevice = 1533c046 source dp SST;
SinkDevice   = 1938c748 sink dp;
```

Here 1533c046 is the TE device serial number and dp stands for display port (vs hdmi or usbc) technology. The optional fourth parameter is used to differentiate devices (more on this later).

To see device info for a SourceDevice device type:

```
TSI -d sourcedevice -devinfo -ll
```

If no errors message are output, you are good to go with this TE (Test Equipment) device. The device info and licenses are listed. Repeat with all devices connected. Note that if the device is open in the UCD Console application, you will get an error.

You generally run TSI.exe from a Command Prompt window with the working directory set to the scripting directory. Scripts are generally run as follows:

```
TSI -r scriptfile.txt
```

The command line *-r* command is used to run scripts in text files.

Queries and small scripts can be executed from the command line:

```
TSI -h -l -d sourcedevice -devinfo -ll
```

Here *-h* will print help, *-l* will print devices connected, *-d* will open device sourcedevice (defined in init.tsi), *-devinfo* will list device information, and *-ll* will list device licenses.

You need to be aware that changing the state of a device via Configuration Items, CI's, can take time and scripting commands calls are asynchronous: So make ample use of the *-waiton*, *-waitonsignal* commands (see 7.3.4 Running TSI.EXE). You can also run

```
-c TSI_W_SCRI_DELAY=1000
```

to sleep for 1000 milliseconds.

7.3.1 Defining test equipment devices

init.tsi may contain the log file name when a time-stamped log file is NOT desired:

```
logfilename=C:\Users\Sam\documents\test_log.txt
```

init.tsi contains the device names (or aliases) and are of the form:

```
DeviceName = 1813C261 source_sink dp TEXT;
```

Here DeviceName is the name you use when opening it (typically SourceDevice or SinkDevice), *1813C261* is the device serial number, *source_sink* specifies the sink and source capability (UDC-400 device designation) and *dp* states that the device uses display port technology (see section 7.3.7 Devices). The fourth TEXT parameter is optional and can be used to differentiate devices that have the same first three parameters: **This is very rarely needed.** TEXT must be a string that can be randomly selected from the native device name (NOT the scripting name) and must be **unique** to that device name.

Typing :

```
tsi.exe -l
```

will output something like:

```
Port Reference Sink (SST, HDCP 2.2)
Dev1 UCD-323 [1533C046]: DisplayPort Reference Source (MST - 2 streams, HDCP
Dev2 UCD-323 [1533C046]: DisplayPort Reference Source (SST, HDCP 2.3)
Dev3 UCD-301 [1938C748]: DisplayPort Reference Sink (SST, HDCP 2.2)
Dev4 UCD-301 [1938C748]: HDMI Reference Sink (HDCP 2.2)
Dev5 UCD-301 [1938C748]: HDMI, DisplayPort (SST), SPDIF Reference Sink
Dev6 TSI Virtual Device

Dev0 = 1533c046 sink dp;
Dev1 = 1533c046 source dp MST;
Dev2 = 1533c046 source dp SST;
Dev3 = 1938c748 sink dp;
Dev4 = 1938c748 sink hdmi;
Dev5 = 1938c748 sink hdmi_spdif;
```

The top-half lines are the device names and bottom-half lines are the scripting names.

Notice that the first three parameters for Dev1 and Dev2 are the same for the scripting names. Note also that the scripting device definitions for Dev1 and Dev2 have the added fourth parameters "MST" and "SST" and that these text strings are present in the device names. Note that "MST" is present in device name Dev1 and the string "SST" is not and visa versa.

The names can be edited for convenience:

```
MyUDC340sink = 1823c395 sink usbc;
MyUDC340source = 1823c395 source usbc;
```

Device definitions can also be given as command line parameter in conjunction with the -d command for opening a TE device:

```
-d "MyUDC340sink = 1823c395 sink usbc"
```

The above two parameter command would define and open device MyUDC340sink. It has the same effect as if it had been included in file *init.tsi*. Note that quotes are needed around the second command line parameter as it must contain spaces.

7.3.2 Commands

Commands are of the form KVE (KeyValueExpression):

```
-c Key=Value.Expression;
```

Note that input to the scripting engine are in the form of one or two command line parameters: This means that if there are spaces in a parameter it must be with quotes. For example

```
- ci_openfile_write "C:\Users\Kari Hall\Documents\TestTSI\Edid.bin"
```

must have quotes because the second command line parameter has a space within it.

Key can be a configuration item (CI) write capable command from this manual (such as -c *TSI_W_USBC_CABLE_CONTROL=1*) which is used to write a value to a CI. When Key is "*.read*" we wish to read a CI Value (such as -c .read=TSI_R_USBC_DP_ALT_MODE_STATUS). When Key is "*.loadtd*" we want to run a built in test (such as -c .loadtd=RunCRCTest.td). *.td file are test files that can be exported from the UCD Console application.

For example, to disconnect CC lines, you would use the following command:

```
-c TSI_W_USBC_CABLE_CONTROL=6;
```

.Expression (optional) selects the bits of *Value* that you want to test or modify for the configuration item in question.

Expressions for commands that WRITE to configuration items are dot-separated keywords of the form:

```
.startbit.number1.length.number2
```

where the first *number1* is the first bit of the sequence and *number2* is the length in bits of the sequence. The sequence must be in the range 0...31.

For example, to change the color space, one of the KVE's in a chain the might be:

```
-c TSI_PG_CUSTOM_TIMING_FLAGS=1.startbit.11.length.4;
```

The above KVE sets the *TSI_PG_CUSTOM_TIMING_FLAGS* configuration item bits 11-12-13-14 to 1 (0001), which is color space YCbCr 4:4:4. Note that the configuration item value will be read and bits 11-12-13-14 will be modified and the resulting value will be written back to the device for that configuration item.

Configuration items requiring more than 32 bits can ONLY be written using array variables.

```
-var Params=20,1      #This is an array with two values
-c TSI_PG_PREDEF_PATTERN_PARAMS=Params.startbyte.0.lengthbytes.8
```

The above example writes 8 bytes or 2 unsigned integers (20 and 1) to the TSI_PG_PREDEF_PATTERN_PARAMS configuration item. startbyte and lengthbytes must be divisible by 4 and lengthbytes must be at least 4.

Commands of the form:

```
-c .read=Value;
```

are used to read the value of a configuration item. The value read is placed in the global variable *lastread*. For example

```
-c .read=TSI_R_USBC_DP_ALT_MODE_STATUS;
```

will read the value of CI TSI_R_USBC_DP_ALT_MODE_STATUS, set its value into global variable *lastread* and if debug mode is 2 or 3 (set via -*o* 2 for example) extended formatted output the will be printed.

Commands of the form:

```
-if .read=TSI_R_USBC_DP_ALT_MODE_STATUS.bitson.0.and.startbit.8.length.4.eq.3;
```

form boolean expressions (1 for true and 0 for false) for -*if .read* commands. Boolean expressions are formed when a relational operator (*.eq. .gt.* Or *.lt.*) and a number or variable are added to the end. -*if .read expressions* can be chained via the logical *.and.* operator.

Expressions for *.read* commands are dot-separated modifier keywords (*bytes, at, bitson, bitsoff, startbit, length and gt, lt, eq*) and integer values (or variables):

- *.bytes.* selects the maximum number of bytes to be read with ARRAY_XX types. It must come right after the CI (Value) and should be divisible by 4.
- *.at.* selects the CI via an index when reading ARRAY_XX types. It must come right after *.bytes.* if present and right after CI (Value) if *.bytes.* is not present.
- *.bitson.1.22* is true (1) when bits 1 and 22 are set. This is a boolean expression.
- *.bitsoff.1.22* is true (1) when bits 1 and 22 are not set. This is a boolean expression.
- *.and.* is a logical operator used to chain boolean exppressions.
- *.eq*. *.lt*. and *.gt* are relational operators for “equal”, “less than”, or “greater than”.

Example:

```
-if .read=TSI_R_USBC_DP_ALT_MODE_STATUS.bitson.0.and.startbit.8.length.4.eq.4;
```

The above command reads the usbc alt-mode status configuration item and will be true if bit 0 is set and bits 8-11 have a value of 4 (“D”: DisplayPort v1.3 2 lanes + USB 3.1 (USB Type C cable)).

Example:

```
-saveread true #TSI_R_DPRX_ERROR_COUNTS is emptied upon read and is 16 bytes
-loops lane=4
    -c .read=TSI_R_DPRX_ERROR_COUNTS.at.lane;
    -pr "Number of errors for lane " -prvar lane -prapp " is " prvar lastread
-endloops
-saveread false
```

The above command reads the error counts for 4 DP lanes. -*saveread* caches the 16 byte read value so subsequent *.read*=TSI_R_DPRX_ERROR_COUNTS.*at.*lane will access the cached value.

A new global variable *lastreadarray* has been added that makes the above example easier:

```
-c .read=TSI_R_DPRX_ERROR_COUNTS
-loops lane=lastreadarray    #Size of lastreadarray
    -pr "Errors for lane " -prvar lane -pr " are " -prvar lastreadarray[lane]
-endloops
```

Example:

```
-c .read=TSI_VERSION_TEXT.bytes.1024;
```

The above command reads the version information (a maximum of 1024 bytes) and prints it out when output mode is 2 or 3 (set via -o 2 or -o 3).

7.3.3 Defining command macros

The *macros.tsi* text file is automatically read if found.

It should contain definitions of user-defined commands of the form (a mix of KVE's and macros delimited by semi-colons all on the SAME line):

```
MyMacro=Key0=Value0.Expression0;MyMacroA;Key1=Value1;MyMacroB;
```

Here *MyMacro* is the new macro being defined, *MyMacroA* and *MyMacroB* are other macros, *Key0* and *Key1* are configuration items or the reserved scripting tokens *.read* or *.loadtd*. *Value0* and *Value1* are the values that are to be read or written for the configuration items. Macros and KVE's must be delimited with semi-colons. In scripts macros in may be executed as:

```
-c MyMacro
```

An example of macros file is:

```
# this line is a comment and will be skipped by the macro definition parser.
# 2lanes macro: Disconnect CC lines, set roles for 2 lanes, make delay of
# 200ms and reconnect.
# "→" means no newline character in actual script file.

2lanes = TSI_W_USBC_CABLE_CONTROL=6; TSI_USBC_DP_ALT_MODE_SETUP=0x80000008;→
         TSI_W_SCRI_DELAY=200;TSI_W_USBC_CABLE_CONTROL=7;

# 4lanes macro: Disconnect CC lines, set roles for 4 lanes, make delay of
# 200ms and reconnect.

4lanes = TSI_W_USBC_CABLE_CONTROL=6; TSI_USBC_DP_ALT_MODE_SETUP=0x80000012;→
         TSI_W_SCRI_DELAY=200; TSI_W_USBC_CABLE_CONTROL=7;

#set source timing colorspace and pattern.

setcolor_to_YCbCr444 = TSI_PG_CUSTOM_TIMING_FLAGS=1.startbit.11.length.4;
apply = TSI_W_PG_COMMAND=4;
set_timing_color_pattern = TSI_W_PG_PREDEF_TIMING_SELECT=3;→
setcolor_to_YCbCr444; TSI_W_PG_PREDEF_PATTERN_SELECT=10; apply;
```

Macros example to set source device to two lane DP Alt Mode and check that it succeeded:

```
-d SourceDevice       #source device from init.tsi
-o 1                  #output level 1
-timeout 20000        #time to wait for -waiton below
# Set DP Alt Mode for UCD-340. This is a comment
-pr ""                #a print new line command
-pr "-----------------------"
-pr "DP Alt Mode 2 lane Test"
-pr "-----------------------"
-pr "Setting Source into 2 lane DP Alt-mode..."
-c Source_2lanesOn    #run macro to set source into 2 lane DP Alt-mode

-d SinkDevice         #sink device from init.tsi
-pr "Setting Sink into 2 lane DP Alt-mode..."
-c 2lanes             #run macro to set sink into 2 lane DP Alt-mode
-waiton ?2lanesModeOn #wait until ?2lanesModeOn is true (a macro)
-pr "Verifying DUT is in 2 lane DP Alt-mode..."
-c ?2lanesModeOn -ontrue -pr "Test PASSED" -pr "Test FAILED" #print results
-pr ""
```

7.3.4 Running TSI.EXE

The program can be run from the Windows command line or from scripting languages such as python. The program is executed as follows:

```
> tsi.exe [command] [target]
```

When executed without any parameters, help will be printed (*-h*).

See the Table below for commands and targets.

Command	Target	Description
-runforever	"directory name"	Keep running and watching a "selected" directory for scripts to run.
-initfile	"file name"	Name and location of init file (default init.tsi).
-logfilename	"file name"	Output log file base name (including path) that is used for time stamping.
-o		Output level (0 – 3).
-err	"stop" or "continue"	Action on test fail (stop or continue).
-l		List connected devices.
-ll		List licenses (a device must be open).
-devinfo		Device information.
-openfile	"file name"	Open file for printing to.
-openfileapp	"file name"	Open file for printing to. Append.
-closefile		Close file for printing to.
-pron		Printing to screen mode on.
-proff		Printing to screen mode off.
-pr	"text"	Print text on new line.
-prapp	"text"	Print text. Append.
-prlastread	"text"	Print text and CI from last .read=Value expression.
-prvar	name name+=number name-=number name*=number name/=number	Print value of variable (loop or global variable) appended to the previous line. Number may be an integer or variable name. Operators may produce decimal output.
-prvarabs	See -pvar above	Same as-pvar above but prints absolute value of variable.
-prvarbits	number or variable	Print value of variable (loop or global variable) appended to the previous line as bits (0s or 1s).
-prvarhex	number or variable	Print value bytes of variable (loop or global variable) appended to the previous line as hex.
-prtimer	"text"	Print text and time in seconds between calls to -timerstart and -timerstop.
-prstr	string variable name	Print string for string variable (defined via -str name).
-prstrapp	string variable name	Print string for string variable. Append.
-d	device name	Select and open device. The device name can be one listed in init.tsi or can be of the form -d "1234c678 source usbc" and must contain space an enclosed in quotes.
-devclose	device name	Close device.
-t		List source timings.
-p		List source patterns.
-m	file name or macro definition	Load macro definitions from a file (default is macros.tsi). Load macro as a command line parameter.
-testparam	key=value	Change a parameters of a test to be run where target is of the form key=value and key is a configuration item.
-saveread	true or false	Save result of next .read=TSI_XXX for subsequent reads.
-c	key=val.exp; or macro	Execute commands.
-r	"file name"	Execute commands from a file.
-xquery	"file name"	Script file to be run when running external program via the -x command to query a state and terminate launched program when script sets global variable failedtests=0.
-x	program name	Launch program (*.exe or *.bat) and continue on close. If a script has been set via -xquery, its return value is monitored

		and theprogram terminated when 0.

-xfeedback	Full path and name of application.	Supplies the full path and name of application that is to be used to handle operator feedback when running tests that require them.
-if	.read=val.exp or n1.RO.n2.LO.n3… or s1.eq.s2 or s1.ne.s2	Conditionally run commands between paired -if -else (optional) -endif commands. n1, n2 etc. are variables or numbers. RO is a relational operator (.eq. .ne. .lt. .gt.). LO is a logical operator (.and. .or.). s1 and s2 are string variables or character strings wiithin single quotes.
-else		Marks the end of the true block of an -if command.
-endif		Marks the end of an -if command.
-yesno	"text"	Prints "text" and asks the user to enter “Y” for yes and “N” for no (works as a truth value for -ontrue).
-pause	"text"	Prints a user message followed by '... press Enter to continue.' and continues when user presses Enter.
-input	"text"	Prints message followed by "Type answer press Enter to continue". Input can be number, variable name or a string. The number, variable value or string is set to global variable xreturn (if number or variable) or sreturn (if string).
-shortcut	"text"	Prints message and waits for shorcut key input. Define global variable array keyshortcuts (e.g. keyshortcuts=72,80,75,77) to define shorcut values of interest. Pressed key value is set to global variable keyshortcut if it is in array keyshortcuts, else it is set to -1.
-var	Name=Val Name=V0,V1,… Name=VA,,VZ Name=Val*Size Name+=Val Name-=Val Name*=Val Name/=Val Name>>Val Name<<Val Name=v1.RO.v2.and.v3	Define or update a global variable or array. Val/V0,V1... may be integers or variable names. Expressions of the form .startbit.number.length.number may be appended to the number or variable to get a number based on a bit range. Variables created using an initializer list (Name=V0,V1,…; Name=VA,,VZ; Name=Val*Size) will be named Name[0], Name[1] with values V0, V1,... and the variable Name is the list SIZE. The << and >> are shift-left and shift-right by Val. Values within braces can be variables and can be nested: -*var* Test=Skip[bool[1]],Skip[bool[0]]. Variables can also be used with two indexes such as Name[i][j] (see restrictions below). The last example will result in a 1 or 0 for true or false. v1, v2 etc. are variables or numbers. RO is a relational operator (.eq. .ne. .lt. .gt.).
-varparam	Name=Value Name=Value0,Value1,… Name=ValueA,,ValueZ Name=Value*Size Name=copy() Name=clear()	Define or update a parameter variable or array (these are only used with -xquery at present). For arrays variables generated will be Name[0],Name[1],... and their values will be Value0, Value1,… Values above can be numbers or numeric variables but NOT other -varparam variables. When Target is "copy()" all -varparam variables will be copied to -var variables. When Target is "clear()" all -varparam variables will cleared. -varparam variables are truly global and are a means of comunicating variables between different script runs as in the case of using -xquery.
-varbytesrev	Variable	Reverse bytes of global variable so that least significant byte comes first.
-str	sName="text" "sName=text" "sName='text'" “sName=Val” “sName=Val1/Val2” “sName=Val1*Val2” “sName=Time()” “sNameA+=text” “sNameA+=sNameB” "Name='text1','text2'"	Define a string variable or array of strings. Target must have quotes if spaces are present. Arrays are delimited by single quotes and commas and name becomes a numerical variable with the number of strings in the array. "text" is plain text, Val,Val1,Val2 are numeric variable names or numbers, Time() will create a timestamp.
-loops	number of loops or Name=Number	Starts looping. Name can be used as a nested variable in -c commands. Name starts at zero when loop starts. (VarName

	Name=VarName	is a variable name.)
-break		Jumps out of current loop.
-continue		Causes subsequent commands up to the -loops pairing -endloops to be skipped.
-endloops		Ends looping (with respect to preceding -loops command).
-skip		Skip, no operation (can be used with -ontrue).
-exit	"text"	Exit scripting and print text.
-ontrue		If preceding .read with expression was true, run the next command, else run that following it.
-timeout	time milliseconds	Timeout for options that require them.
-timerstart		Start timer.
-timerstop		Stop timer.
-waitonsignal	time milliseconds	Wait until video or audio signal is detected or it has timed out (a device must be open).
-waiton	.read=val.exp;	Wait on a .read=Value.Expression to become true or time out (a device must be open).
-pic	"file name"	Capture and save a picture to file (a device must be open).
-picset	"file name"	Set source pattern generator custom image.
-blanktop	Number of pixels	Skip vertical pixel rows from top when using -pic command.
-blankbot	Number of pixels	Skip vertical pixel rows from bottom when using -pic command.
-frames	Number of frames	Frames to capture when using -pic command.
-wcrc		Also save a *.crc32 file when saving a picture to file via -pic.
-checkcrc	"crc file name"	Check the crc values of the image created via -pic and the value in the target crc file name.
-beginhtms		Begin new html test report (a device must be open).
-endhtml		End html test report (a html file must be open).
-redid	filename.ecd	Read edid data from file to device (a device must be open).
-wpdosink	file name	Write device sink PDO data to file (a device must be open).
-rpdosink	file name	Read sink PDO data from file to device (must be open).
-wpdosource	file name	Write device source PDO data to file (a device must be open).
- rpdosource	file name	Read source PDO data from file to device(must be open).
-ci_openfile_write	base file name	Open binary file to write configuration item data to.
-ci_closefile_write		Close binary file to write configuration item data to.
-ci_use_header	"false" or "true"	Use header when writing configuration item data. Default is true.
-ci_writedata	configuration item	Configuration item which is to be saved to file.
-h		List help for each command.

-runforever is used to start tsi.exe in "watchdog" mode. In this mode tsi.exe keeps running and watching a target "watchdog" directory for scripts to run. This directory should be empty or only contain scripts to be run and deleted (as they are run). Files can be dropped or copied (via a javascript or python program, for example) into the directory. All opened devices remain open after each script executes: Subsequent scripts still need to open devices via the ***-d*** *devicename* command (which will be fast since the device is already open from the previous script). Any script that executes an ***-exit*** command causes tsi.exe to stop running. Scripts in the "watchdog" directory are run as they are found **except** if a script "**exit.txt**" is found which is run immediately. Before running each script, the script file is deleted (eaten). Script output should not be directed to the "watchdog" directory" as this will cause execution to terminate via a scripting error as tsi.exe tries to run it. All files dropped into the "watchdog" directory for scripts to run must have the *-logfilename* command so results can be directed to their proper output directories.

-initfile is the path/name of the init file to initialize devices (default is init.tsi located in the working directory). This command is optional.

-logfilename is the log file base name (including path) that is used for time stamping. For example, if the target is C:\Users\Karih\Documents\LogBin\Runtests.txt the log file name would look something like Runtests_20190516103934_log.txt and located in the C:\Users\Karih\Documents\LogBin directory. When in *-runforever* "watchdog" mode, each script file will be parsed and the *-logfilename* command MUST be found and

the first occurance will be used. When NOT in "watchdog" mode, this command is optional but must be a command line parameter (i.e. not in an external file or macro) if it is to be used. This will override any logfilename found in init.tsi.

-o specifies the amount output desired: *-o* 0 selects minimum output. *-o* 1 is used for debugging.
-o 2 is used for extended output (such as edid values).
-o 3 is a combined *-o* 1 and *-o* 2.
With *-o* 2 and *-o* ,3 .reads=TSI_XX; to the following CI's produce extended output:

TSI_R_USBC_CABLE_STATUS,
TSI_R_USBC_POWER_SINK_RDO,
TSI_USBC_PWR_LOCAL_SINK_PDO,
TSI_R_USBC_PWR_REMOTE_SINK_PDO,
TSI_USBC_PWR_LOCAL_SOURCE_PDO,
TSI_R_USBC_PWR_REMOTE_SOURCE_PDO,
TSI_R_USBC_POWER_SOURCE_PDO,
TSI_R_USBC_DP_ALT_MODE_STATUS,
TSI_DEVICE_FIRMWARE_INFO,
TSI_VERSION_TEXT,
TSI_R_PG_PREDEF_PATTERN_NAME,
TSI_EDID_TE_INPUT,
TSI_EDID_TE_OUTPUT,
TSI_DPRX_DPCD_DATA,
TSI_R_USBC_IDO_TABLE.

-err specifies the whether to continue or stop when a test or query fails.

-l lists the connected devices.

-ll lists the licenses for an open device.

-devinfo lists device information.

-openfile opens a file for printing to. Statements that produce printed ouput will print to the open file: *-pr*, *-prapp*, *-prvar*, *-prlastread* and *-ll*. Only one file can be open for printing to at one time: Opening a new file while a file is open will cause it to be closed.

-openfileapp is the same as *-openfile* above but printed output will be appended to file.

-closefile will close file for printing to. The global variable *timestampfiles* will be used to determine if the filename supplied by *-openfile* or *-openfileapp* will be timestamped upon closing.

-pron sets print to screen mode on. Print commands are printed to screen and regular output.

-proff sets print to screen mode off. Print commands are not printed to screen and regular output. This can be used with in conjuction with *-openfile.*

-pr prints the target text on a new line.

-prapp prints the target text appended to previous line (no carriage return).

-prlastread prints the target text followed by the value of the last CI (Configuration Item) value read.

-prvar prints the variable value (loop or global variable) appended to the previous line. The + - * and / operators can be used to modify the output. These operators can be used to produce decimal output.

-prvarabs is same as *-prvar* above but prints absolute value of variable.

-prvarbits prints the variable as binary value (loop or global variable) as bits appended to the previous line.

-prvarhex prints bytes of variable (loop or global variable) appended to the previous line as hex.

-prstr prints string for string variable (defined via -str name).

-prstrapp prints string for string variable (defined via -str name) appended to current line.

-d selects and opens (if it is not already open) a device by name. The name can be listed in init.tsi or be of the form "1234c678 source usbc" or "DeviceName = 1234c678 source usbc". Use the *-l* command to get the correct form for each device. With the second naming convention, the device name will be 1234c678sourceusbc which can be used with subsequent calls to *-d*. With the third naming convention the device name would be DeviceName.

-devclose closes the device (hardware device is closed).

-t lists source device timings.

-p lists source device patterns.

-m loads macro definitions from a file (the default *macros.tsi* is also loaded if found first). It may also be defined via a command line parameter. For example: *-m* Wait5seconds=TSI_W_SCRI_DELAY=5000;
Important: *This is read in as it is parsed so commands that rely on these macros must come after. Macros with the same name overwrite those defined earlier.*

-testparam *c*hanges a parameter of a test to be run where target is of the form key=value and key is a configuration item. After tests are run all *-testparams* values are cleared.
An example could be:
-testparam TSI_CRC_FRAMES_TO_TEST=100
-*c* .loadtd=CRCSingleFrameVideoStabilityUDC340.td

-saveread can be set to "true" or "false" so that subsequent read operations can access the reasults of a previous read operation with the same CI (configuration item). This is vital for array reads that empty results buffers or registers on read. The CI is noted and subsequent reads for the same CI will use the cached value until a different CI is read or a *-saveread false* option/target is executed.

-c executes an expression of the form *Key=Value.Expression*; or a Macro.

-r executes commands from a file. These are equivalent to console program command line parameters: Each space not within parentheses marks the start of a new command or target. Commands and targets must be on the SAME line.
-c .read = TSI_R__XXX would be interpreted as four command line parameters while
-c ".read = TSI_R__XXX" would be interpreted as two command line parameters.

-xquery is used to define the script file to be run when running external program via the *-x* command to query a device state and to terminate the launched program based on the return value. If the value returned from running the script is 0 the launched process is terminated. Set global variable *failedtests* to 0 in script file to terminate the program launched via the *-x* command. See Example 5 below.

-x is used to launch an external program or run a batch file. If no script file is supplied via the *-xquery* command it waits until the process finishes to continue. If a script file has been set via the *-xquery* command, the script is run at intervals defined via the current timeout (can be set via the *-timeout* command). If the value returned from running the script is 0 (Set global variable *failedtests* to 0 in script.) the launched process is terminated, otherwise the lauched program needs to terminate itself. The return value is stored in the global variable *xreturn*. See Examples 4 and 5 below.

-xfeedback is used to handle operator feedback when running tests that require them. The target is the full path and name of application used to handle operator feedback (see "Operator feedback during test execution" in this manual). This needs to be run before the test:
-xfeedback "C:\TSIX\tsi.exe"
-*c .loadtd*=test443.td

-if conditionally runs commands between paired *-if* - *else* (optional) and - *endif* commands.

It's *target* can be a .read=CI.expression.

Here CI is a configuration item.
The expression part uses the CI value read in from the TE device to evaluate a boolean value (0 or 1).
Boolean values are obtained via an optional selection operator pair (.startbit. .length.) and a relational operator (.eq. .ne. .lt. .gt.) or just boolean operator (.bitson. .bitsoff.). Examples:
-if .read=TSI_USBC_DP_ALT_MODE_SETUP.startbit.2.length.3.eq.1
-if .read=TSI_USBC_DP_ALT_MODE_SETUP.eq.658876
-if .read=TSI_USBC_DP_ALT_MODE_SETUP.bitson.1.5

The *target* can also be of the form n1.RO.n2.LO.n3…

Here n1, n2 etc. are variables or numbers. RO is a relational operator (.eq. .ne. .lt. .gt.) and LO is a logical operator (.and. .or.).
The expression is evaluated from left to right:
Ordering of the form ***-if*** n0.eq.1.and.n1.ne.10.or.n2.ne.1.and.n3.lt.5.or.n4.eq.0 makes things easy to understand.
An example is:

```
 -if timing.lt.0.or.timing.gt.Tmax.and.timing.ne.10
    #Do Something
-endif
```

The *target* can also be of type: s1.eq.s2 or s1.ne.s2.

Here s1 and s2 are character strings within single quotes ('text') or string variables defined via the -str command.
Examples are:

```
-if 'UCD-323'.eq.DeviceNames[s]
    #Do something ...
-endif
-if 'UCD-323'.ne.DeviceNames[s]
    #Do something ...
-endif
```

-else marks the end of the true block of an *-if* command. It is optional.
An example is:

```
 -if timing.lt.0.or.timing.gt.Tmax.and.timing.ne.10
    #Do something when "timing.lt.0.or.timing.gt.Tmax.and.timing.ne.10" is true
-else
    #Do something else ...
-endif
```

-endif marks the end of commands for its paired *-if* command. If there is an *-else* between a pair of *-if* - *endif* commands *-else* marks the end of the true block and - *endif* marks the end of the false block. If there is not an *-else* command between a pair of *-if* - *endif* commands - *endif* marks the end of the true block.

-yesno prints "text" and asks the user to enter "Y" for yes and "N" for no (also works as a truth value for *-ontrue*).

-pause prints a user message followed by "... press Enter to continue.". Execution continues when user presses Enter.

-input prints message followed by "Type answer press Enter to continue". Input can be number, variable name or a string. The number, variable value or string is set to global variable *xreturn* (if number or variable name) or *sreturn* (if input was not number or variable name).

-shortcut prints message and waits for shorcut key input. Define the global variable array keyshortcuts (e.g. keyshortcuts=72,80,75,77) to define shorcut values of interest. The pressed key value is set to global variable *keyshortcut* if it is in array keyshortcuts, else it is set to -1. Set output level to 1 (-o 1) to discover pressed key values. These can then be set into variable array keyshortcuts.

See example "Navigate timings and patterns from the keyboard" below.

-var defines or updates a global variable.

Variable names must start with an alphabetic character and may also contain numerical digits as well as the underscores ("_") and brackets ("[" and "]"). Subsequent statements with the same variable name update the existing global variable. Variable names are case insensitive.

Global variables may be incremented or decremented via the += and -= operators. Also the *= and /= operators are available for scaling variables.

Built-in variables are also available (see descriptions below). Variables defined are global except *-loops* variables, which are nested (they vanish after *-endloops* is executed). If there is a nested and global variable of the same name, the nested variable is evaluated in expressions.

Variables can be created using an *initializer list:*
-var Name=V0,V1,V3
-var Name=VA,,VZ (values VA,VA+1…VZ-1)
-var Name=Value*Size (values Value,Value,Value …)
Name[0], Name[1] , Name[2] ... will have values V0,V1,V3 etc.
Name itself is a variable and is the list size.
Right-hand side values can be numbers or variables.
Values within braces can be variables and can be nested:
-var Test=Skip[bool[1]],Skip[bool[0]].
Variables can also be used with *two* indexes such as Name[i][j] with the *restriction* that the variables within braces do not themselves have braces.

Numbers or variables used can have an appended expression of the form.startbit.number.length.number to select a bit range that is transformed into a number:
-var Vab=255.startbit.0.length.8

Variables can also be used as boolean values and evaluated via boolean expressions:
-*var bitsareon=234.bitson.1.3.5*.
The value assigned to *bitsareon* would be 1 or 0 if the expresion result is true or false.

Boolean expressions may be chained using the *.and.* operator:
-*var test=234.bitson.1.and.p.gt.5.and.255.startbit.0.length.8.eq.4* .
The value of test will be 1 if *234.bitson.1, p.gt.5 and 255.startbit.0.length.8.eq.4* are all true.

Simple arithmetic can be performed via the +=, -=, *=, /=, <<, >> operators.
-var Name+=3 will increment the value of Name by 3.
-var Name<<1 will left shift the bits of the value of Name by 1.

-varparams define or update a parameter variable or array (these are only used with *-xquery* at present).

-varparam variables are truly global and are a means of comunicating variables between different script runs as in the case of using *-xquery.*

For a single numeric variable the *target* is of the form: Name=Value.
For an array of numeric variables *target* is of the form:
Name=Value0,Value1,... or Name=ValueA,,ValueZ or Name=Value*Size.
Name will be a numeric parameter variable whose value is the number of variables in the array.
Variables generated will be Name[0],Name[1],... and their values will be Value0, Value1,…
Values above can be numbers or numeric variables but NOT other *-varparam* variables.

When *target* is "copy()" all *-varparam* variables will be copied to *-var* variables.
When *target* is "clear()" all *-varparam* variables will cleared.

-varbytesrev reverses bytes of a *-var* variable so that least significan byte comes first.

-str defines string variables.

Note that the entire *target* is a command line parameter which means it must be contained within double quotes if there are spaces anywhere within the *target*!

-*str* Name=StringValue defines a single string variable.

StringValue is a string (defined within single quotes) or string variable defined via -*str* command.
-*str* Name='UCD-323'
-*str* "Name='Hello world!' "
-*str* Name=VarString (a string variable)

-*str* Name=Numeric
Numeric is a number or numeric variable (defined via -*var* command).
-*str* Name=15
-*str* Name=VarNumeric (a numeric variable)

-*str* Name=numberA*numberB
-*str* Name=numberA/numberB
(numberA and numberB are numbers or numeric variables)

-*str* Name=time() (a timestamp will be generated)

-*str* Name='Value1','Value2','Value3', … (defines an array of string variables)

Name will be a numeric variable with the number of strings in the array.
Name[0] will have Value1, Name[1] will have Value2, ...

-*str* "sa='time()','V','V*3','V/10','t','Hello World'"
V is a numeric variable and t is a string variable.
sa[0] will be something like 2021:01:25:16:09:50
sa[1] will be something like 100
sa[2] will be something like 300

Use the += operator to add to a string variable.

-*str* "NameA='Goodbye ' "
-*str* "NameB='cruel World!' "
-*str* NameA+=NameB (value now is 'Goodbye cruel World!'

-loops starts a loop.

-loops SzLoops
-loops s=SzLoops
SzLoops above is the number of loops to execute and s is a local variable starting starting at 0 and ending with value SzLoops-1.

Looping is to the next (same nesting level) -*endloops* command. Loops can be nested and there must be the same number of -*loops* and -*endloops* commands.

-break jumps out of current loop.

-continue causes subsequent commands up to the -*loops* pairing -*endloops* to be skipped.

-endloops marks the end of a loop of the same nesting level started by the previous ***-***loops command option.

-skip is a no-operation and can be used with the -*ontrue command*.

-exit exits scripting and prints a message.

-ontrue is a conditional operator: If the preceding .read with expression (required) was true, the next command (-*option* target) is run, else that following it is run. It cannot be nested.

-timeout will set the timeout time in milliseconds (default is 10000) for commands that require them.

-waitonsignal will wait until video and/or audio signal is detected on the selected device (a device must be open) or the timout period (target value in milliseconds) has elapsed.

-waiton will wait for a *.read=Value.Expression* to become true or a time out (a device must be open). Can be used to wait for a DUT to be plugged in, for example. Its main purpose though is making sure CI's to be tested are in a proper state for reading as setting CI's are asynchronous (see 1.2.4.1 EXAMPLES below).

-pic is used to capture and save a picture to a file. A device sink must be open and a DUT source must be connected for this to succeed. The format output is a lossless bitmap (*.bmp). Once executed the -*blanktop* and -*blankbot* values will be set to zero.

-picset is used to set a source device custom image from a file.

-blanktop can be called before -*pic* to skip vertical pixel rows from the top when using the -*pic* command. The -*pic* command will reset the -*blanktop* value to zero.

-blankbot command can be called before -*pic* to skip vertical pixel rows from the bottom when using the -*pic* command. The -*pic* command will reset the -*blankbot* value to zero.

-frames command can be called before -*pic* to capture a number of frames. The frame number (starting at 1) is appended to the -*pic* file name when the number of frames is greater than 1. The -*pic* command will reset the -*frames* value to 1.

-wcrc Also save a *.crc32 file when saving a picture to file via -pic. The CRC (cyclic redundancy value) is just a value that represents the data in question. If any bit in an image changes, its CRC value will change. The text file (*.crc32) consists of three unsigned integer values: a version number, the value 1 and the actual CRC value.

-checkcrc Check the crc values of the image created via -*pic* and the value found in the target *.crc32 file. Executing -*checkcrc* will also cause the the -*pic* command to output a *.crc32 file. The target file name must have the *.crc32 extension. The global variable *failedtests* will be incremented if the CRC values are different.

-beginhtml begins new html test report that receives output ONLY for tests (***-c*** *.loadtd*=test.td). A device must be open. If the device is changed via a -*d Dev* command, the file will be closed (equivalent to -*endhtml*). The html output file will be of the form <*devsink*>*_20181008105224_log.html*, where <*devsink*> is the device name.

-endhtml ends the current open html test report (a html file must be open).

-redid reads sink edid data from a file of the same form created with Unigraf EDID Editor (with a .ECD suffix.) and sets it on the open sink device. Each line begins with a field of the form "$08 - " followed by hex text characters: $08 - 54 c1 36 40 4c 34 32 30. Semicolons at the beginning of a line are comments and ignored.

-wpdosink writes device sink PDO data (see *TSI_USBC_PWR_LOCAL_SINK_PDO*) to a file.

-rpdosink reads sink PDO data from a file and writes it to the open device.

-wpdosource writes device source PDO data (see *TSI_USBC_PWR_LOCAL_SOURCE_PDO*) to a file.

-rpdosource reads source PDO data from a file and writes it to the open device.

-ci_openfile_write opens a target file in binary mode so that configuration item data can be saved in binary form. This will happen with subsequent calls to *-ci_writedata TSI_XXX* and a *-c .read=TSI_XXX* (both commands must be run) where TSI_XXX is the configuration item whose data we want to save. The target is the base name and location of the ooutput binary file. For example, if the target is C:\Users\Karih\Documents\LogBin\ PDEvents.bin, the binary file name would look something like PDEvents_20190516103934.bin and located in the C:\Users\Karih\Documents\LogBin directory.

-ci_closefile_write closes the binary file opened with *-ci_openfile_write*.

-ci_writedata *TSI_XXX* specifies the configuration item (*TSI_XXX*) for which data will be saved via a subsequent *-c .read=TSI_XXX command.* A configuration item write file must be open. A target of "" will disable data output to the file.

-ci_useheader "false" or "true" (default) specifies whether a header is to be used when writing configuration item data to a binary file. A configuration item write file must be open. When no header is used, the data is written as a binary blob. The header has the following binary format: At the beginning of the file, bytes 0:3 are the four letters "TSIX", bytes 4:5 are a 16 bit version number , and bytes 6:7 are reserved. For each configuration item the following data is written: The CI value is 4 bytes (unsigned integer), the size of the CI binary data blob is 4 bytes (unsigned integer), the timestamp is 8 bytes (unsigned 64 bit integer), and finally the "data" is written as a binary blob.

-h lists options and targets.

Built-in variables are also available:

failedtests is the number of tests that failed. **This is also the value returned from TSI at exit.**
lastread is the value of the last .read=TSIX_XXX operation.
lastreadarray is the array of values of the last .read=TSIX_XXX operation.
size*lastread* is the size (number of integers read) of the last .read=TSIX_XXX operation.
lasttestpassed (0 for failed and 1 for passed) is the value of the last .loadtd=filename.td test.
lasttestskipped is incremented when a test is run but skipped.
errorcount is the number of errors encountered that do not cause scripting termination. It is is incremented when a picture cannot be captured or saved successfully, when a test cannot be run, or when *-err* is set to "continue" and a scripting error occurrs.
xreturn contains the value returned from a process launched via the *-x* command or the return value from *-input* command when input value is number or variable name.
sreturn is a string global variable and contains the return value from *-input* command when input value is a string.
keyshortcut contains the input value from the *-shortcut* command.
erroroccurred contains the error number when something goes wrong. When it is positive we have a scripting event and when it is negative we have a TSI Engine SDK error. The currently available scripting errors are 1 (Script error.) and 2 (-waiton timed out..). More will be added as the needed. See example below.
errormessage is a string variable (see *-str* above) and contains the error message when something goes wrong and is set at the same time as *erroroccurred* above.
timestampfiles is used to determine if the file name associated with *-ci_open_file_write* is timestamped when *-ci_closefile_write* is executed and the file is saved.
timer contains the time in milliseconds between calls to ***-****timerstart* and ***-****timerstop* calls. It is updated on calls to ***-****timerstop*.
true has value 1.
false has value 0.

7.3.4.1 Examples

Example 1. Looping color spaces, patterns and timings :

```
-timout 20000 #Wait on video signal
-d SourceDevice #defined in init.tsi
-c .read=TSI_R_PG_PREDEF_TIMING_COUNT;
-var SzT=lastread -pr "Timeing count is " -prvar SzT
-c .read=TSI_R_PG_PREDEF_PATTERN_COUNT;
-var SzP=lastread -pr "Pattern count is " -prvar SzP
-var dpix=10;  #delta value for pixel width and height tests
-var dhz=2;    #delta value for frequency
-var false=0
-var true=1
-var CsSkip=false,true,true # CsSkip[0]=0 CsSkip[1]=1 CsSkip[2]=1
-loops cs=3           #three color spaces
-loops p=SzP          #p starts at 0 for patterns
    -loops t=SzT      #t starts at 0 for timings
        -if CsSkip[cs].eq.1.and.p.ne.10
            -continue #only want to run non-RGB tests for pattern 10
        -endif
        -d SourceDevice
        -c TSI_W_PG_PREDEF_TIMING_SELECT=t;
        -c TSI_W_PG_PREDEF_PATTERN_SELECT=p;
        -if p.ne.10.and.cs.ne.0.or.cs.eq.0
            -c setcolorSpace_RGB         #a macro
        -else
            -if cs.eq.1
               -c setcolorSpace_YcbCr420 #a macro
            -else
               -c setcolorSpace_YcbCr422 #a macro
            -endif
        -endif
        -c TSI_W_PG_COMMAND=1;       #apply command
        -c .read=TSI_PG_CUSTOM_TIMING_HACTIVE;
        -var wdpix=lastread;         # lastread is a global variable
        -c .read=TSI_PG_CUSTOM_TIMING_VACTIVE;
        -var htpix=lastread;
        -c .read=TSI_PG_CUSTOM_TIMING_FIELD_RATE;
        -var hz=lastread;
        -var hz/=100; # scaling to work with TSI_R_INPUT_FREQ below
        -pr "" -pr "Timing " -prvar t -prapp " " -prvar wdpix
        -prapp "x" -prvar htpix  -prapp " @ " -prvar hz/10 -prapp "Hz"
        -prapp "\"  -prapp "Pattern " -prvar p -prapp " "
        -c .read=TSI_R_PG_PREDEF_PATTERN_NAME.bytes.256
        -d SinkDevice
        -var wdpixmin=wdpix; -var wdpixmin-=dpix;
        -var wdpixmax=wdpix; -var wdpixmax+=dpix;
        -waiton .read=TSI_R_INPUT_WIDTH.gt.wdpixmin.and.lt.wdpixmax;
        -var htpixmin=htpix; -var htpixmin-=dpix;
        -var htpixmax=htpix; -var htpixmax+=dpix;
        -waiton .read=TSI_R_INPUT_HEIGHT.gt.htpixmin.and.lt.htpixmax;
        -var hzmin=hz; -var hzmin-=dhz;
        -var hzmax=hz; -var hzmax+=dhz;
        -waitonsignal 50000  #wait for video signal timeout is 5 secs
        -if .read=TSI_R_INPUT_WIDTH.eq.wdpix
            -if .read=TSI_R_INPUT_HEIGHT.eq.htpix
                -if .read=TSI_R_INPUT_FREQ.gt.hzmin.and.lt.hzmax
                    -pr " ----> PASS"
                -else
                    -prlastread "Frequency ----> FAILED: Value Read = "
                -endif
            -else
                -prlastread "Height ----> FAILED: Value Read = "
            -endif
        -else
            -prlastread "Width ----> FAILED: Value Read = "
        -endif
    -endloops  #t=SzT
-endloops      #p=SzP
-endloops      #cs=3
```

Example 2. Manipulating Power Data Objects (also see Setting Power Data Objects below):

```
#Basic variables
-var true=1 -var false=0
-var pdo_disabled=0 -var pdo_fixed=1 -var pdo_variable=2 -var pdo_battery=3

#define default values for SOURCE and SINK as pdo_fixed
-var types=pdo_fixed,pdo_fixed,pdo_fixed,pdo_fixed,pdo_fixed

#define default loading values for SOURCE and SINK
-var PO[0]=3000,5000,100       #list PO[0] defines variables PO[0][1]=3000 ...
-var PO[1]=3000,9000,100
-var PO[2]=3000,12000,100
-var PO[3]=3000,15000,100
-var PO[4]=3000,20000,100
-var PO[5]=3000,20000,100
-var PO[6]=3000,20000,100

#Variables Source to select which operations to do
-var do_pdos=0                  #list of source POs to change. Default none.
-var disable_pdos=0             #list of source POs to disable. Default none.

#variable test used to select operations below
-var test=1

#Redefine those PDOs that we want to change
-if test.eq.0                   #Change PDO 1 to type variable
    -var do_pdos=1,             #comma IS important because this is a list
    -var types[1]=pdo_variable  #NO comma we are just changing a list value
    -var PO[1]=9000,9000,8000   #commas because we are redefining a list
    -var disable_pdos=2,        #want to disable PDO2
-endif
-if test.eq.1
    -var do_pdos=1,2,3,4        #reset PDOs 1,2,3,4 to defaults
-endif

-d sourcedevice                 #select device
-var is_source=true             #required by -r SetPDOs.txt

#define variables pdo_sel, pdo_type, and pdo_values needed by -r SetPDOs.txt
-if do_pdos.gt.0                        #change PDOs
    -pr "Setting source side power data objects ..."
    -loops p=do_pdos                    #do_pdos is the number of do_pdos
        -var pdo_sel=do_pdos[p]         #which PO to change
        -var pdo_type=types[pdo_sel]    #what type do we want it to be
        -var pdo_values=PO[pdo_sel][0],PO[pdo_sel][1],PO[pdo_sel][2],0 #Set
        -r SetPDOs.txt                  #add changes
    -endloops
-endif

-if disable_pdos.gt.0                   #disable PDOs
    -loops p=disable_pdos
        -var pdo_sel=disable_pdos[p]
        -var pdo_type=pdo_disabled
        -r SetPDOs.txt
    -endloops
-endif

-if do_pdos.gt.0.or.disable_pdos.gt.0
    -c TSI_W_USBC_INITIAL_ROLE=2;#initial port role DFP
    -c TSI_USBC_PWR_COMMAND=1    #Set the source power objects
-endif

#Variables Sink to select which operations to do
#Values with commas to enable (even with one value).
-var do_pdos=0      #list of sink POs to change. Default is none:
-var disable_pdos=0 #list of sink POs to disable. Default is none.
```

```
#Redefine values that differ from default BELOW
-if test.eq.0
    -var do_pdos=1,
    -var types[1]=pdo_variable
    -var PO[1]=9000,9000,8000
    -var disable_pdos=2,
-endif
-if test.eq.1
    -var do_pdos=1,2,3,4    #reset PDOs 1,2,3,4 to defaults
-endif

-d sinkdevice           #select device
-var is_source=false    #required by -r SetPDOs.txt

#define variables pdo_sel, pdo_type, and pdo_values needed by -r SetPDOs.txt
-if do_pdos.gt.0
    -pr ""
    -pr "Setting sink side power data objects ..."
    -loops p=do_pdos
        -var pdo_sel=do_pdos[p]         #which PO to change
        -var pdo_type=types[pdo_sel]    #what type do we want it to be
        -if pdo_type.eq.pdo_fixed
            -var pdo_values=PO[pdo_sel][0],PO[pdo_sel][1],0
        -else
            -var pdo_values=PO[pdo_sel][0],PO[pdo_sel][1],PO[pdo_sel][2]
        -endif
        -r SetPDOs.txt
    -endloops
-endif
#define variables pdo_sel and pdo_type needed by -r SetPDOs.txt
-if disable_pdos.gt.0
    -loops p=disable_pdos
        -var pdo_sel=disable_pdos[p]
        -var pdo_type=pdo_disabled
        -r SetPDOs.txt
    -endloops
-endif
-if do_pdos.gt.0.or.disable_pdos.gt.0
    -c TSI_W_USBC_INITIAL_ROLE=1;  #initial port role UFP
    -c TSI_USBC_PWR_COMMAND=2      #Setsink power objects
-endif

-c RECONNECT        #a macro to reconnect
-c WAIT_5s          #a macro to wait 5 seconds

#Check power contract
-pr ""
-if .read=TSI_R_USBC_POWER_STATUS.bitson.2
  -pr "*** Power contract established. "
-else
  -pr "*** NO power contract. "
-endif
```

Example 3. Setting Power Data Objects (called by Example 2 above):

```
#Example file SetPDOs.txt
#Variables used:
# is_source     Must set is_source=1 if source power objects, else is_source=0
# pdo_sel       Select the power object via variable pdo_sel (must be 0-6)
# pdo_type      Select the type via variable pdo_type (must be 0-3)
#               To disable a power object set its type to 0: -var pdo_type=0
# pdo_values    Set via pdo_values var: pdo_values=3000,5000,125,0 for ex.

#Some script level error checking
-if is_source.ne.0.and.is_source.ne.1
    -exit "Exiting from Script: Variable is_source needs to be 0 or 1."
-endif
-if pdo_type.lt.0.or.pdo_type.gt.3
    -exit "Exiting from Script: pdo_type needs to be 0, 1, 2 or 3."
-endif

#Set the values using CIs
-if is_source.eq.1    #if source power object
    -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_SELECT=pdo_sel
    -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_TYPE=pdo_type
    -if pdo_type.eq.1 #fixed supply
        -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_CURRENT=pdo_values[0]
        -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_VOLTAGE=pdo_values[1]
        -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_PEAK_CURRENT=pdo_values[2]
     -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_FIXED_SUPPLY_BITS_25_TO_29=pdo_values[3]
    -endif
    -if pdo_type.eq.2 #variable supply
        -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_CURRENT=pdo_values[0]
        -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_VOLTAGE=pdo_values[1]
        -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_MIN_VOLTAGE=pdo_values[2]
    -endif
    -if pdo_type.eq.3 #battery supply
        -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_POWER=pdo_values[0]
        -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_MAX_VOLTAGE=pdo_values[1]
        -c TSI_USBC_PWR_LOCAL_SOURCE_PDO_MIN_VOLTAGE=pdo_values[2]
    -endif
-else                 #if sink power object
    -c TSI_USBC_PWR_LOCAL_SINK_PDO_SELECT=pdo_sel
    -c TSI_USBC_PWR_LOCAL_SINK_PDO_TYPE=pdo_type
    -if pdo_type.eq.1 #fixed supply
        -c TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_CURRENT=pdo_values[0]
        -c TSI_USBC_PWR_LOCAL_SINK_PDO_VOLTAGE=pdo_values[1]
       -c TSI_USBC_PWR_LOCAL_SINK_PDO_FIXED_SUPPLY_BITS_25_TO_29=pdo_values[2]
    -endif
    -if pdo_type.eq.2 #variable supply
        -c TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_CURRENT=pdo_values[0]
        -c TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_VOLTAGE=pdo_values[1]
        -c TSI_USBC_PWR_LOCAL_SINK_PDO_MIN_VOLTAGE=pdo_values[2]
    -endif
    -if pdo_type.eq.3 #battery supply
        -c TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_POWER=pdo_values[0]
        -c TSI_USBC_PWR_LOCAL_SINK_PDO_MAX_VOLTAGE=pdo_values[1]
        -c TSI_USBC_PWR_LOCAL_SINK_PDO_MIN_VOLTAGE=pdo_values[2]
    -endif
-endif
```

Example 4. Executing a *.exe and a *.bat file examples:

```
-x "C:\Windows\system32\notepad.exe" #run notepad and wait until it is closed

-x "cmd.exe /c hello.bat"            #run a batch file on Windows
```

Example 5. Cable testing setup. Here a UDC-400 could be used for testing cables by running a script and connecting the DP RX input and DP TX output recepticles via a DP cable when prompted via a dialog program. As soon as a connection is detected the dialog disappears. The tester can also terminate the dialog program manually via the Ok or Cancel buttons.

```
# Launch ImageRequest.exe dialog program.
# Run script  testcableconnection.txt every 1000 ms.
# to detect if cable has been plugged in.
# When cable is plugged in testcableconnection.txt will return 0
# and ImageRequest.exe dialog program will be terminated.
# Use -xquery before -x to set script.

-timeout 1000                        #set polling time in ms to run script
-xquery testcableconnection.txt      #file to test if cable connected
#Launch ImageRequest.exe dialog program.
-x "ImageRequest.exe -i Pic.bmp -c "Connect Cable"  -bc Cancel -ba Ok"
-timeout 10000                       #Restore timeout to default value

#Below is testcableconnection.txt file
-var failedtests=1    #this is the global variable returned from script
-d SourceDevice       # Open Source device
-o 0                  # Set output to "No debug messages"
-pr " "
-c .read=TSI_R_DPTX_HPD_STATUS       #Read TX HPD status.
-var Connected=lastread.bitson.0     #if bit 0 is set, we have a connection.
-if Connected.eq.1
    -pr "DP Cable is CONNECTED!"
    -var failedtests=0       #cable is connected so set return value to 0
-endif
```

Example 6. Print message based on a .read=Value.Expression:

```
-c .read=TSI_R_USBC_DP_ALT_MODE_STATUS.bitson.0
-ontrue -pr "Alt Mode On." -pr "Alt Mode Off."
```

Example 7. Print if type-C cable is in straight or flipped orientation:

```
 -c .read = TSI_R_USBC_CABLE_STATUS.bitsoff.1;
 -ontrue -pr "Strait orientation" -pr "Flipped orientation"
```

Example 8. Looping:

```
-loops 2 #outer loop
    -pr "outer loop cmd 0"
    -pr "outer loop cmd 1"
    -loops 3 #inner loop
        -pr "   inner loop cmd 0"
        -pr "   inner loop cmd 1"
    -endloops #inner loop
    -pr "outer loop cmd 2"
```

```
-endloops
```

Example 9. Looping with variables:

```
-var Times=3,5,6,8,10
-var Pats=0,5,10
-loops i=Pats #outer loop 3 times
    -c TSI_W_PG_PREDEF_PATTERN_SELECT=Pats[i];
    -loops j=Times #inner loop 5 times
       -c TSI_W_PG_PREDEF_TIMING_SELECT=Times[j];
    -endloops
-endloops
```

Example 10. Using if-else-endif:

```
-if .read = TSI_R_USBC_CABLE_STATUS.bitsoff.1
       -pr "Straight orientation"
-else
    -pr "Flipped orientation"
-endif
```

Example 11. Reading bits from MSA data (ARRAY_U32):

```
-c TSI_W_DPRX_MSA_COMMAND=1
-c TSI_R_DPRX_MSA_STREAM_COUNT=1
-c .read=TSI_R_DPRX_MSA_DATA.at.3           #read the fourth integer
-var Hactive=lastread.startbit.0.length.16  #select first 16 bits of lastread
-pr "H Active = " -prvar HActive
```

Example 12. Writing arrays to a configuration item:

```
-var Params=20,1
TSI_PG_PREDEF_PATTERN_PARAMS=Params.startbyte.0.lengthbytes.8
```

Example 13. Read edid values from a DUT sink and output to a binary file:

```
-o 0
-d SourceDevice  #Open device
-ci_openfile_write ".\Edid.bin"      #binary output file base name
-ci_writedata TSI_EDID_TE_OUTPUT     #CI whose data we want to output
-ci_useheader false                  #output pure binary blob
-c .read=TSI_EDID_TE_OUTPUT;         #See reference manual
-ci_writedata ""                     #reset
-ci_closefile_write                  #close the file
```

Example 14. Check if -waiton command timed out:

```
# ErrorOccurred is a global variable that is set whan -waiton times out
# errormessage is a global string variable that is set whan -waiton times out
-waiton .read=TSI_R_HDCP_2X_STATUS.bitson.0 # check if HDCP 2.X is active
-if ErrorOccurred.ne.0               # ErrorOccurred is a global variable
    -pr "Error value is " -prvar ErrorOccurred
    -prstr errormessage      # errormessage is a global string variable
-endif
```

Example 15. Navigate timings and patterns from the keyboard using arrow keys:

```
-o 0

-var true=1
-var false=0

#navigation keys can be found via sequence:
# -O 1 #debug output mode
```

```
# -shortcut "hit a key"
# the key hit value is printed
-var keyup=72
-var keydn=80
-var keylh=75
-var keyrh=77
-var shortcuts=keyup,keydn,keylh,keyrh  #This is required for -shorcut command

-d sourcedevice

-c .read=TSI_R_PG_PREDEF_TIMING_COUNT;
-var SzT=lastread -pr "Total timing count is " -prvar SzT
-var Tmax=SzT
-var Tmax-=1

-c .read=TSI_R_PG_PREDEF_PATTERN_COUNT;
-var SzP=lastread -pr "Total pattern count is " -prvar SzP
-var Pmax=SzP
-var Pmax-=1

-var timing=5
-var pattern=0

-var SzLoops=1000000
-loops SzLoops

    -var timeold=timing
    -pr "Give timing."
    -input ""
    -var timing=xreturn
    -if timing.eq.-1
        -exit "Exiting ...."
    -endif
    -if timing.lt.0.or.timing.gt.Tmax
        -exit "Exiting ... illegal timing."
    -endif

    -pr ""
    -pr "Selected timing is " -prvar timing
    -pr "Selected pattern is 0"
    -c TSI_W_PG_PREDEF_TIMING_SELECT=timing;       #Select timing
    -c TSI_W_PG_PREDEF_PATTERN_SELECT=pattern;     #Select pattern
    -c TSI_W_PG_COMMAND=1;                         #Set them

    -pr ""
    -pr "Use the up and down arrow keys to iterate through timings."
    -pr "Use the left and right arrow keys to iterate through patterns."
    -pr "Any other key to input another timing."
    -pr ""
    -loops SzLoops
        -o 1
        -shortcut ""
        -o 0
        -var key=keyshortcut
        -var patold=pattern
        -var timeold=timing
        -if key.eq.keyup   #Up
            -var timing+=1
        -endif
        -if key.eq.keydn   #Down
            -var timing-=1
        -endif
        -if key.eq.keyrh   #Right
            -var pattern+=1
        -endif
        -if key.eq.keylh   #Left
            -var pattern-=1
        -endif
        -if patold.eq.pattern.and.timeold.eq.timing
            -break
        -endif
        -if pattern.gt.Pmax
            -var pattern=0
        -endif
        -if pattern.lt.0
            -var pattern=Pmax
```

```
        -endif
        -if timing.gt.Tmax
            -var timing=0
        -endif
        -if timing.lt.0
            -var timing=Tmax
        -endif
        -pr "timing is " -prvar timing
        -pr "pattern is " -prvar pattern
        -c TSI_W_PG_PREDEF_TIMING_SELECT=timing;          #Select timing
        -c TSI_W_PG_PREDEF_PATTERN_SELECT=pattern;        #Select pattern
        -c TSI_W_PG_COMMAND=1;                            #Set them

    -endloops
-endloops
```

Example 16. E-marked cable VBUS Current Handling Capability:

```
# ErrorOccurred is a global variable that is set whan -waiton times out
# errormessage is a global string variable that is set whan -waiton times out

-waiton .read=TSI_R_HDCP_2X_STATUS.bitson.0 # check if HDCP 2.X is active
-if ErrorOccurred.ne.0              # ErrorOccurred is a global variable
    -pr "Error value is " -prvar ErrorOccurred
    -prstr errormessage      # errormessage is a global string variable
-endif

-d SinkDevice
-c SET_CABLE_ORIENTATION_STRAIGHT
-c InitialPortRoleDFP # EMarker can be read only in DFP mode
-c CONNECT_CC_LINES
-c .read=TSI_R_USBC_CABLE_STATUS

-saveread true #Since we want to read an array
-var id=0
-o 2   #Want to print out E-marked cable data
-c .read=TSI_R_USBC_IDO_TABLE
-o 0

-var sz=sizelastread  #How much data was read
-pr ""
-if sz.gt.0 # Size is zero if cable is unplugged or UCD340 not in DFP mode
   -loops id=sz
       -c .read=TSI_R_USBC_IDO_TABLE.at.id
       -if id.gt.2
           -var vdo=lastread
           -var vdo<<25
           -var vdo>>30
           -if 1.eq.vdo
               -pr "Note: Cable eMarker has 3A VBUS Current Capability"
           -else
               -pr "Note: Cable eMarker has 5A VBUS Current Capability"
           -endif
       -endif
   -endloops
-else
    -pr "E-Marker data can't be read."
    # Check cable connection
    -if .read=TSI_R_USBC_CABLE_STATUS.startbit.0.length.1.eq.0
       -pr "Check that Cable is connected between UCD-340 and DUT."
```

```
    -endif
    # Check DFP mode is active
    -if .read=TSI_R_USBC_ROLE_STATUS.startbit.0.length.1.eq.0 # UFP detected
       -pr "Check that UCD340 is in DFP mode."
    -endif
-endif

-saveread false  #Since we no longer want to read array TSI_R_USBC_IDO_TABLE
```

7.3.5 Output

Output is directly to the console window and possibly to an output file.

Output to a file is automatic when scripts are run from a file:

```
-r file.txt
```

The output file will be of the form: *file_20181008105217_log.txt*

Init.tsi may contain the log file name when a time-stamped log file is not desired:

```
logfilename=C:\Users\Sam\documents\test_log.txt
```

The output log file name can also be a command line parameter:

```
-logfilename=C:\Users\Sam\documents\test_log.txt
```

Extra html output for tests (see 7.3.6 Running Tests below) can be started via *-beginhtm* and ended via *-endhtml*.

If the device is changed via a *-d* command or if another *-beginhtml* command is executed, the file will be closed (equivalent to *-endhtml*).

The html output file will be of the form *<devsink>_20181008105224_log.html* where *<devsink>* is the device alias name.

7.3.6 Running Tests

Tests are of the form:

```
-c .loadtd=filename.td;
```

filename.td is a test that has been saved (from the UCD Console program for example). The global variable *lasttestpassed* is set to 1 for passed and 0 for failed.

Some tests require operator feedback: the DUT (device under testing) needs to be set into a defined state. Tsi.exe can be run as an external application for this. A typical test requiring feedback would be run as follows:

```
-c TSI_TS_OF_MODE=1 #see Operator feedback during test execution in the manual
-xfeedback  "C:\TSIX\tsi.exe" -c .loadtd=test443.td
```

When tsi.exe calls for operator feedback it will launch another instance of tsi.exe with various parameters (see "*Operator feedback during test execution"* in this manual). The first parameter is op=number which defines the operator feedback id. The parameters may look like: op=9;res_x=1280;res_y=800;res_frate=60000;res_bpp=12;exit_proceed=9; Tsi.exe takes these parameters (except for op and exit_proceed) and creates global variables with these names. exit_proceed is used for the instance return value. Tsi.exe then creates a single commad of the form: -r OperatorFeedBackRequest_ID_9.txt (for op=9). Tsi.exe expects to find this file and proceeds to run it. It will exit with the value "exit_proceed" and the instance running the test will proceed. An example of a feedback file for op=2 could be:

```
#Operator Request op=2;res_x=640;res_y=480;res_frate=60000;exit_proceed=2
-o 0
-var testrun=0 #Set to 1 for testing this script
#below for testing
-if testrun.eq.1
    -var op=2
    -var res_x=640
    -var res_y=480
    -var res_frate=60000
    -var exit_proceed=2
-endif
#above for testing

#actual script to use as operator feedback below
-d SourceDevice #select device
-var timing=0   #timing variable set below
-r FindPredefinedTiming.txt #calculate timing id from res_x, res_y,  res_frate
-c TSI_W_PG_PREDEF_TIMING_SELECT=timing
-pr "Predefined timing = " -prvar timing
-prapp " "   -prvar res_x
-prapp " x " -prvar res_y
-prapp " @ " -prvar res_frate*0.001 -prapp "Hz"

-c TSI_W_PG_PREDEF_PATTERN_SELECT=9;
-c TSI_W_PG_COMMAND=4; #Set_PG_Output_On
-c WAIT_3s
```

When -r OperatorFeedBackRequest_ID_2.txt is run it will output a regular log file of the form: OperatorFeedBackRequest_ID_2_20181102131212_log.txt which can be used for checking and debugging.

Some devices such as the UDC-400 have both inputs and outputs. Here tests in loopback mode (same device output to same device input) requiring operator feedback can be run within the same process. Scripts for operator requests as described above must be available. In this case OperatorFeedBackRequest_ID_2_20181102131212_log.txt type logs will not be created.

```
-c TSI_TS_OF_MODE=2 #see Operator feedback during test execution in the manual
-c .loadtd=test443.td
```

7.3.7 Devices

Devices need to be listed in the file init.tsi. A method for creating this file from the command line is:

```
tsi.exe -l > init.tsi
```

The user can edit init.tsi and name the devices as desired (default names are dev0, dev1, etc.).

A physical device may have one or more “devices” in different roles. Devices are of the form: *Name = SerialNumber Role Technology*

Examples:

Devices such as the UDC-400 are listed as:

```
Dev0 = 1813c261 source_sink dp;
```

Devices such as the UDC-422 are listed as:

```
Dev0 = 1813c261 source_sink hdmi;
```

Devices such as the UDC-340 are listed as:

```
Dev0 = 1823c395 sink usbc;
Dev1 = 1823c395 source usbc;
```

Devices such as the UDC-323 are listed as:

```
Dev0 = 1642c158 sink dp;
Dev1 = 1642c158 source dp;
Dev2 = 1642c158 sink hdmi;
Dev2 = 1642c158 source hdmi;
```

Devices such as the UDC-301 are listed as:

```
Dev0 = 1636c147 sink dp;
Dev1 = 1636c147 sink hdmi;
Dev2 = 1636c147 sink hdmi_spdif;
```

Devices such as the UDC-1 are listed as:

```
Dev0 = 1636c147 sink lvds;
```

8 TESTSTAND INTEGRATION

This version of TSI introduces integration functions for National Instrument's TestStand. This section describes how to use these functions to configure the TE device, set TSI test parameters and run TSI tests.

8.1 Integration functions

TestStand integration is done by creating 4 new functions. Please notice that these functions are exported with C++ name mangling enabled in order to have the function parameter types show up in TestStand automatically.

8.1.1 TSI_TST_Init

ClientVersion 12, and higher *No license requirements*

```
void __cdecl TSI_TST_Init
(
    char *iSetupScript,
    bool &oErrors,
    long &oErrorCode,
    char oErrorMsg[1024]
);
```

Description

This function is expected to be used to initialize TSI library when using it through TestStand. The function will run the given script through the iSetupScript parameter. If the script contains errors, or the device(s) being used are not available oErrors is set to true, oErrorCode will have a machine readable error code, and the oErrorMsg will be set to a human-readable error message.

Parameters

iSetupScript

Pointer to a NULL terminated string. The string content can be either the setup-script directly, or a reference to a file that contains the setup script.

Important: *For details on the scripting, please refer to 8.2 TSI Scripts in TestStand.*

oErrors

Reference to a variable that receives error flag. The value is set to "true", if the function failed, or "false" if the function succeeded.

(Continued...)

(...Continued)

oErrorCode

Reference to a variable that receives an error code. Error codes are non-zero negative values. Zero (and positive values) indicate no error result.

oErrorMsg

Pointer to the first byte of a 1024 byte character array which receives a human readable error message if the function failed. If there were no errors, the resulting error message is empty.

Result

If the function succeeds, the error flag is set to “false”, Error code and Error message are cleared. If the function fails, the error flag is set to “true”, Error code and Error message variables are set to indicate the error.

8.1.2 TSI_TST_RunTest

ClientVersion 12, and higher *No license requirements*

```
void __cdecl TSI_TST_RunTest
(
     char *Device,
     char *ConfigScript,
     int TestID,
     bool &Passed,
     bool &Error,
     char oErrorMsg[1024],
     char oReportText[64000]
);
```

Description

The function runs the test indicated by TestID value. The test is executed on a hardware unit indicated by the Device value. Before the test is started, any script given in ConfigScript is executed. If the test passes, the Passed is set to true, and otherwise false. If the function encounters any errors, the Error flag is set to true. If Error flag is set to true, the oErrorMsg will contain a human readable error message. oReportText will contain test log as human readable text data.

Parameters

Device

Pointer to a string containing the name of the device which is to run the test. The device names are user defined in the initialization script at the time TSI_TST_Init was called.

ConfigScript

Pointer to a test configuration script. If the device is already configured this script can be empty. The script can be directly passed as string, or it can be a file-name from which the script is loaded.

(Continued...)

(...Continued)

TestID

Indicates which test to run. Please refer to chapter 6 for information about tests and their ID values.

Passed

Reference (pointer) to a boolean variable that will be set to “true” if the test was completed with pass status. If the test failed the variable will be set to “false”.

Error

Reference (pointer) to a boolean variable that will be set to “false” if the test completed successfully. If there were errors during the test execution, this variable set set to “true”.

oReportMsg

Pointer to a character buffer of 1024 characters. The buffer will receive a human readable error message in the event that “Error” was set to “true”.

oReportText

Pointer to a character buffer of 64000 characters. The buffer will receive test log printout as human readable ASCII text data.

Result

The function will process the given configuration script. If the script contains errors, the error flag will be set to “true”, and report message will contain message about the error in the script. If the script was processed without errors, it will continue to run the test indicated by the TestID and Passed, Error, oReportMsg and oReportText variables will be set according to the outcome of the test.

8.1.3 TSI_TST_RunScript

ClientVersion 12, and higher *No license requirements*

```
void __cdecl TSI_TST_RunScript
(
     char *Device,
     char *iScript,
     bool &Error,
     char oErrorMsg[1024],
     int *oIntTable,
     int iMaxCount
)
```

Description

This function is used to run scripts on a device without running any tests. In addition to test configuration scripts, the scripts passed to this function are also capable of returning integers from the script.

Parameters

Device

Pointer to a string containing the name of the device which is to run the test. The device names are user defined in the initialization script at the time TSI_TST_Init was called.

iScript

Pointer to a string containing the script to be processed. The script can be directly passed as string, or it can be a file-name from which the script is loaded.

Error

Reference (pointer) to a boolean variable which will be set to “true” if the script contained errors. If the script was completed without errors, this variable will be set to “false”.

oErrorMsg

Pointer to a character buffer of 1024 bytes which will receive a human readable error message if the script contained errors.

oIntTable

Pointer to a table of integers that will receive any values returned by the script. The returned values are placed into the output table in the order they were read.

iMaxCount

Size of the integer table passed in oIntTable, as count of integers.

Result

The given script is executed. Any values read are placed into the table given in oIntTable. In case of error in the scripts, the Error and oErrorMsg variables are set accordingly.

8.1.4 TSI_TST_Clean

ClientVersion 12, and higher *No license requirements*

```
void __cdecl TSI_TST_Clean()
```

Description

Release all allocated resources and close all devices. This function should be called when the TSI functions are not going to be used by the test program.

Result

All resources allocated by TSI are released and devices are closed.

8.2 TSI Scripts in TestStand

This section defines how to write scripts to be used with the TSI_TST_Init, TSI_TST_RunTest and TSI_TST_RunScript functions. As each of these functions has a slightly different use for the scripts, the scripts also are slightly different.

Please notice, that TSI has a two-levels of scripts. The scripts used with TestStand integration are considered simplistic, and it offers read/write operations for the configuration items. It has no flow control mechanisms or other advanced features – these types of features are expected to be provided on a higher level language (like within a test sequence in TestStand).

8.2.1 TestStand script syntax

The script language syntax follows these rules.

- A statement consists of a key and an associated value.
- The key and value must be on the same line of text.
- There must be an equal sign (‘=’) between key and value.
- There must be a semi-colon character (‘;’) after the value.
- If either key or value contain spaces, it must be enclosed with single-quote marks (‘).
- White-space characters that are not part of key nor value are ignored, and therefore can be used to format the script to be more readable.

Below are some examples of valid script lines. The first line assigns a filename into a configuration item, and the second one assigns a numeric value. You can find the used configuration item definitions elsewhere in this document.

```
·····TSI_LOG_FILE·=·'C:\My·TSI·Logs\Log.txt';···
·····TSI_W_SCRI_DELAY·=·1000;
```

Important: *The (‘·’) characters are white-space characters, such as spaces or tabs.*

8.2.2 Writing a script for TSI_TST_Init

The intention of an initialization script is to create a known environment for the rest of the test sequence and make test station configuration easier. If there is a need to distribute a TestStand test sequence to multiple stations, it can be very useful to put the initialization script into a file, and refer to that file from the TestStand test sequence. This way, each test station can be configured to use the appropriate hardware by modifying the script file with notepad, or some other simple text editor utility, while the TestStand test sequence itself remains unchanged.

The script provided for TSI_TST_Init through the iSetupScript pointer is used to define aliases for the devices used in the test sequence.

```
MyDevice = 'UCD-323 [1234C5678]: DisplayPort Reference Source';
```

The above script would define an alias "**MyDevice**". The alias is later used to identify the device by providing it through the *Device* string pointer parameters available in the TSI_TST_RunTest and TSI_TST_RunScript functions. The "**MyDevice**" alias will refer to the device given in the value field – (In this case a UCD-323 device with S/N 1234C5678 being used as a DisplayPort source). Please notice that defining a device alias will also open the device. As a result, it is not possible to define multiple aliases that would require use the same physical hardware.

In addition to selecting a device, it is necessary to also select which interfaces are going to be used. TSI offers selection of an input, and an output. An input and output can be used simultaneously, provided this is supported by the hardware. Currently, this is supported by UCD-400 alone at the moment of writing this document. Below is a script that selects both, an input and an output:

```
MyDevice = 'UCD-400 [1234C5678]: DisplayPort Reference Sink and Source';
.output='DP-1';
.input='DP-1';
```

An output is is selected with a special key "**.output**", and input is selected with another special key "**.input**". The values are expected to contain the name of input or output being selected for use. The "**.input**" and "**.output**" selectors apply to the previous device alias definition. Please do remember that available inputs and/or outputs depend on the type of hardware being used. These special keys (".output" and ".input") are available for use in a script that is provided for TSI_TST_Init, and can't be used in other scripts.

It is also possible to define multiple device aliases: Consider a case where two UCD-323 devices are being used, for example, to test a repeater device. One of the UCD-323 devices would be used as a source device, with the other one acting a sink device. A following script might be used in this case:

```
Source = 'UCD-323 [1234C567]: HDMI Reference Source';
.output = 'HDMI-1';
Sink = 'UCD-323 [4321C876]: HDMI Reference Sink';
.input = 'HDMI-1';
```

This will create two device aliases "**Source**" and "**Sink**", with "**Source**" using the HDMI-1 output port, and "**Sink**" using the HDMI-1 input port.

8.2.3 Writing a script for TSI_TST_RunTest

The intention of script given to this function is to configure a test before running it. The script can also change device settings if necessary.

The script provided for TSI_TST_RunTest is used to configure test parameters. This can be done by assigning each configuration item one by one, or by loading a **TD-file** (Test Description file). The TD-Files can be saved from UCD-Console.

To load a TD-File from a script, use '.loadtd' key, and assign the TD-File name to it. Example:

```
.loadtd = 'C:\my_td_files\Test 1.td';
```

Important: *The TSI_TST_RunTest function will not run the test defined in the TD file. It will load the test settings from it, but it will run the test identified by the TestID parameter.*

Alternatively, the test settings can be done one-by-one by assigning (one or more) configuration items. The following example sets the audio test parameters:

```
TSI_EXPECTED_SAMPLE_RATE = 96000;
TSI_EXPECTED_AUDIO_FREQUENCY = 1000;
TSI_AUDIO_FREQUENCY_TOLERANCE = 1;
TSI_AUDIO_GLITCH_DETECT_TRESHOLD = 250000;
TSI_AUDIO_GLITCHES_ALLOWED = 0;
```

Important: *You can also refer to configuration items by their ID code number, not just their names. This can be useful, in case there is some kind of problem with identifying a configuration item by name.*

8.2.4 Writing a script for TSI_TST_RunScript

The intention of script given to this function is to extend the functionality of the scripts.

The script provided for TSI_TST_RunScript is intended to be used for device control and status checking. The TSI_TST_RunScript has additional parameters that enable 32 bit data words to be returned from the scripts. These scripts can also load TD-Files, same way as scripts provided to TSI_TST_RunTest.

To read 32-bit data and return it out of the script, use '.read' key, and assign the configuration item ID to read. See below for an example:

```
.read = TSI_R_INPUT_WIDTH;
.read = TSI_R_INPUT_HEIGHT;
```

The above script would return two 32-bit integer values through the oIntTable pointer. The 32-bit word at index 0 would contain the current x-resolution, and the 32-bit word at index 1 would contain the current y-resolution.

Any actions based on the returned data must be decided on the TestStand side. This scripting system is very simple, and has no flow control or other advanced features.