TSI

Full Reference Manual

24. November 2016

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 Reference Manual

Document ID

Issue date 24. November 2016

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 1.6 [R3] (TSI.DLL [1.6.3])

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 2.2 Using the TSI API with more detailed information.
- 18.8.2015
  Added descriptions for functions 3.10.1 TSI_REP_BeginLogRecord, 3.10.2 TSI_REP_EndLogRecord and 3.9.4 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.

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

2. COMPONENTS AND FEATURES

This section describes the features and components of the TSI API.

2.1.API DLL

The TSI API implementation is available as a 32-bit DLL and as 64-bit DLL. Naturally, the 64-bit version is only available for 64-bit operating systems.

On 32-bit operating systems (Windows XP, Windows Vista/7/8 x86 versions), the 32-bit TSI.DLL is typically located in the "C:\program files\common files\Unigraf\shared" folder.

On 64-bit operating systems (Windows XP x64, Windows Vista/7/8 x64 versions), the 32-bit TSI.DLL is typically in the "C:\program files (x86)\common files\Unigraf\shared" folder, while the 64-bit TSI.DLL is typically in "C:\program files\common files\Unigraf\shared" folder.

The provided loader will primarily load the TSI.DLL from one of the default locations, and if the DLL file is not found, it tries default OS search paths.

2.1.1.Features

- **Backwards compatibility guaranteed**: New versions of TSI are guaranteed to support all functionality of all previous TSI API versions. This means that application built using older TSI SDK can use TSI DLL from a later version SDK.

  Important: *Engineering builds may introduce features that will not be available and/or might be modified in a following actual release. Engineering builds are always clearly marked as such.*

  Important: *The backwards compatibility starts from the first official production release of TSI, which is version 1.2.*

- **Simplified usage**: The API offers a high level of intelligence built in so that client applications can use less API function calls and avoid using multiple threads for one device.

2.2.Using the TSI API

This API is intended to work as a high level API offering specific features to customers using the existing API interfaces of supported hardware.

Simplified usage

This API does not use handles to refer to devices. Instead, the API commits to using only the selected device. Using multiple devices with this API requires that each device is handled in it's own process.

Thread safety

TSI API is protected against harmful concurrent access.

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 attempts to allow using several versions of low-level APIs per supported hardware, but may require specific versions. The required minimum versions are listed in the "release notes.txt" file that comes in the TSI release package.

Function return values

The primary rules for all functions that use a return value of type TSI_RESULT are the following:

- Negative value is always an error.
- Zero value indicates a generic success.
- Non-zero positive values indicate a generic success and additionally convey information specific to the function that returned the value.

The outcome of the rules is that

- Test for success can, and must be done as follows (or equivalent in your programming language):
  `if(Result >= TSI_SUCCESS) { /* Success */ }`
- Test for failure can, and must be done as follows (or equivalent in your programming language):
  `if(Result < TSI_SUCCESS) { /* Failure */ }`
- `if(Result == TSI_SUCCESS) { /* Specific return check */ }` does not test for success in general, instead it tests if a specific success condition has happened.

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

3. FUNCTIONS

This section describes the TSI client callable functions.

3.1.External functions

The functions in this chapter are delivered as source code.

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 second. 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 succesfully, but the Init function failed, the return value is zero. In this case it will be necessary to call the 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() call was last the Unload 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.API Base level functions

3.2.1.TSI_Init

ClientVersion 1, and higher

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

Synopsis

Initializes the API for use. Calls to Init are reference counted: Clean() must be called equal number of times for correct operation. If 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 Init to ensure compatibility with later versions of the DLL.

Important: *The first call to 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 wanted 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

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

Synopsis

Closes device and releases API resources if the reference count after the function call equals zero. Calls to 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

```
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, 3.3.7 TSI_DEV_Select, 3.3.5 TSI_DEV_GetDeviceCount

3.3.2.TSI_DEV_GetParameterID

ClientVersion 4, and higher

```
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

```
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 is not interested in seeing 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

```
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

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

Synopsis

Enumerates supported Unigraf capture 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 it's 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, 3.3.7 TSI_DEV_Select

3.3.6.TSI_DEV_GetDeviceName

ClientVersion 1, and higher

```
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

ID Value identifying the wanted 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_Select

ClientVersion 1, and higher

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

Synopsis

Selects a capture device to be activated. The intention of this function is to provide client application means to select one device out of many.

Parameters

DeviceID

Identifies the device to open. 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 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

3.4.Input management functions

3.4.1.TSI_VIN_GetParameterCount

ClientVersion 4, and higher

```
TSI_RESULT __stdcall TSI_VIN_GetParameterCount
(
     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 TSI_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

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 TSI_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 TSI_VIN_GetParameterID, 3.4.3 TSI_VIN_GetInputCount

3.4.2.TSI_VIN_GetParameterID

ClientVersion 4, and higher

```
TSI_RESULT __stdcall TSI_VIN_GetParameterID
(
     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

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 TSI_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 TSI_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.

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 TSI_VIN_GetParameterCount, 3.4.3 TSI_VIN_GetInputCount, 3.4.5 TSI_VIN_Select

3.4.3.TSI_VIN_GetInputCount

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_VIN_GetInputCount();
```

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. If no capture device is active, this function will activate capture device with device ID of zero.

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 TSI_VIN_GetInputName, 3.4.5 TSI_VIN_Select

3.4.4.TSI_VIN_GetInputName

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_VIN_GetInputName
(
     TSI_INPUT_ID InputID,
     char *InputNameString,
     unsigned int NameStringMaxLen
);
```

Synopsis

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

Parameters

InputID

ID value of the input to be identified. Valid InputID values range from zero (0) to the number of inputs returned by TSI_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 length for 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 zero.

See Also

3.4.3 TSI_VIN_GetInputCount, 3.4.3 TSI_VIN_GetInputCount

3.4.5.TSI_VIN_Select

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_VIN_Select
(
     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. By default, input number 0 is selected on the active device. If there is no need to change the active input this function call can be omitted completely.

Parameters

InputID

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

Result

If the function succeeds, the return value is a non-zero positive containing flag bits that indicate the data types which can be captured from the input. Please refer to 4.6.2 Data stream type flags for further details.

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

See Also

3.4.3 TSI_VIN_GetInputCount, 3.4.6 TSI_VIN_Enable

3.4.6.TSI_VIN_Enable

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_VIN_Enable
(
     TSI_FLAGS Flags
);
```

Synopsis

Enable Audio/Video input. This function can succeed only if the input status at time of call is disabled.

The Input's enable state works as a gate-keeper for all data capturing: If the input is disabled, no data is being captured nor processed. When the input is enabled, all available data formats are captured.

If no device or audio/video input is selected before calling this function, this function will select the device with device ID of zero, and uses the default input. Please refer to 3.3.7 TSI_DEV_Select and 3.4.5 TSI_VIN_Select for further details.

Parameters

Flags

Obsolete. Any value passed here is ignored, and data formats to be captured are automatically detected and captured as they become available.

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

3.3.7 TSI_DEV_Select, 3.4.5 TSI_VIN_Select, 3.4.7 TSI_VIN_Disable

3.4.7.TSI_VIN_Disable

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_VIN_Disable();
```

Synopsis

Disables the audio/video input. This will stop all audio/video processing in the API.

Result

If the function succeeds, the audio/video input state is set to 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 TSI_VIN_Enable

3.5.Video Preview functions

3.5.1.TSI_VPREV_SetWindowHandle

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_VPREV_SetWindowHandle
(
     HWND 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 enabled. To disable video preview set preview window handle to NULL.

Parameters

Container

Handle to the window to contain the video preview. If a video is already being shown in another window that preview will stop and then start again in the new 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 non-zero positive value to indicate success.

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

3.6.Audio Preview Functions

3.6.1.TSI_APREV_SetWindowHandle

Client Version 3, and later

```
TSI_RESULT __stdcall TSI_APREV_SetWindowHandle
(
     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 audio capture is enabled and audio is received. To disable the preview set preview window handle to NULL.

Parameters

Container

Handle to the window to contain the video preview. If a video is already being shown in another window that preview will stop and then start again in the new window. If this parameter is NULL, then any existing video 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 non-zero positive value to indicate success.

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

See Also

-

3.6.2.TSI_APREV_GetDeviceCount

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_APREV_GetDeviceCount();
```

Synopsis

Returns the number of audio playback devices found from the local system.

Result

If the function succeeds, the return value is a positive value (or zero) indicating the number of available audio playback devices. If the return value is zero, there are no suitable audio playback devices present.

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

See Also

3.6.3 TSI_APREV_GetDeviceName, 3.6.4 TSI_APREV_SelectDevice

3.6.3.TSI_APREV_GetDeviceName

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_APREV_GetDeviceName
(
     TSI_AUDIO_DEVICE_ID PlaybackDeviceID,
     char *PlaybackDeviceNameString,
     unsigned int NameStringMaxLen
);
```

Synopsis

Retrieves a human readable name associated with the given playback device ID.

Parameters

PlaybackDeviceID
: Indicates the audio device to be identified.

PlaybackDeviceNameString
: Pointer to an array of characters that will receive the name of the audio device. The string is guaranteed to be NULL terminated. If the string buffer is not large enough to contain the full name it will be truncated.

NameStringMaxLen
: Number of chars available in the PlaybackDeviceNameString character array. Recommended size is 128 characters or more.

Result

If the function succeeds, the return value is a positive number indicating the number of characters required to hold the device's full name not counting the terminating NULL. If the return value is EQUAL or HIGHER than NameStringMaxLen parameter, it means that the string was truncated.

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

See Also

3.6.2 TSI_APREV_GetDeviceCount

3.6.4.TSI_APREV_SelectDevice

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_APREV_SelectDevice
(
     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, issue device ID negative one (-1).

Parameters

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 non-zero positive value to indicate success.

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

See Also

3.6.2 TSI_APREV_GetDeviceCount

3.7.Test system related functions

3.7.1.TSI_TS_GetTestCount

ClientVersion 3, and higher

```
TSI_RESULT __stdcall TSI_TS_GetTestCount();
```

Synopsis

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

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

3.7.2 TSI_TS_GetTestInfo

3.7.2.TSI_TS_GetTestInfo

ClientVersion 3, and higher

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

Synopsis

Retrieves the test ID values and test names of the test available on the currently selected device.

Parameters

TestIndex

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

ID

Pointer to a TESTI_TEST_ID variable, which will receive the test ID value of 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 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

3.7.1 TSI_TS_GetTestCount, 3.7.3 TSI_TS_GetTestParameterCount

3.7.3.TSI_TS_GetTestParameterCount

ClientVersion 3, and higher

```
TSI_RESULT __stdcall TSI_TS_GetTestParameterCount
(
     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 TSI_TS_GetReqParameterID function in a loop.

Important: *Use of this function is not needed for application that only uses fully known device types.*

Parameters

ID

Identifies the test of which to get the parameter count. This test ID value is retrieved either by using the TSI_TS_GetTestInfo function, or by using a constant value defined in a devices 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

3.7.4 TSI_TS_GetReqParameterID

3.7.4.TSI_TS_GetReqParameterID

ClientVersion 3, and higher

```
TSI_RESULT __stdcall TSI_TS_GetReqParameterID
(
     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

ID

Identifies the test to query.

ParamIndex

Index value ranging from zero (0) to value returnde by a call to TSI_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_PID_MUST_SET	Indicates that the parameter must be set before attempting to run the test. If this bit is not set, then setting the parameter is optional (= it has a default value).
1	TSP_PID_VIRTUAL_GROUP	The parameter is a group identifier. Group identifiers can't be directly set. Instead each group refers to a set of other ID values.

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

3.7.3 TSI_TS_GetTestParameterCount

3.7.5.TSI_TS_Clear

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_TS_Clear();
```

Synopsis

Resets the test system to it's default settings. Please refer to section 4.3 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.

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

4.3 Tests

3.7.6.TSI_TS_SetConfigItem

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_TS_SetConfigItem
(
     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 4.4.2 Configuration items for video format read 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

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

3.7.7 TSI_TS_GetConfigItem, 3.7.9 TSI_TS_LoadConfig

3.7.7.TSI_TS_GetConfigItem

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_TS_GetConfigItem
(
     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

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

3.7.6 TSI_TS_SetConfigItem, 3.7.8 TSI_TS_SaveConfig

3.7.8.TSI_TS_SaveConfig

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_TS_SaveConfig
(
     char *Filename
);
```

Synopsis

Saves the current test system status to a file for later use. 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

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

3.7.9 TSI_TS_LoadConfig

3.7.9.TSI_TS_LoadConfig

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_TS_LoadConfig
(
     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. 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 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

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

3.7.8 TSI_TS_SaveConfig

3.7.10.TSI_TS_RunTest

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_TS_RunTest
(
     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 4.3 for details on available tests. 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. Also, an input must be selected before calling this function. If no input is selected before calling this function, this function will select the default input.

Parameters

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

4.3 Tests

3.7.11.TSI_TS_CaptureReference

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_TS_CaptureReference
(
     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. 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. Also, an input must be selected before calling this function. If no input is selected before calling this function, this function will select the default input.

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

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

3.7.6 TSI_TS_SetConfigItem, 3.7.9 TSI_TS_LoadConfig

3.7.12.TSI_TS_WaitInputSignal

ClientVersion 6, and higher

```
TSI_RESULT __stdcall TSI_TS_WaitInputSignal
(
     unsigned int MaxWait
);
```

Synopsis

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

Parameters

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.

3.8.Misc functions

3.8.1.TSI_MISC_SaveReference

Client Version 4, and later

```
TSI_RESULT __stdcall TSI_MISC_SaveReference
(
     char *FileName,
     unsigned int RefIndex,
     TSI_FRAME_FORMAT_ID FormatID
);
```

Synopsis

Save the current reference frame into a file.

Parameters

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.
2	TSI_FRAME_FORMAT_PPM	Identiefies 24 or 48 effective bits per pixel PPM file type.

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

3.8.2 TSI_MISC_LoadReference, 3.7.11 TSI_TS_CaptureReference, 3.7.8 TSI_TS_SaveConfig

3.8.2.TSI_MISC_LoadReference

Client Version 4, and later

```
TSI_RESULT __stdcall TSI_MISC_LoadReference
(
     char *FileName,
     unsigned int RefIndex
);
```

Synopsis

Load a reference image from a file.

Parameters

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

3.8.1 TSI_MISC_SaveReference, 3.7.9 TSI_TS_LoadConfig

3.8.3.TSI_MISC_SetOption

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_MISC_SetOption
(
     TSI_OPTION_ID OptionID,
     int OptionValue
);
```

Synopsis

Get and set option value. The function will set the new option value, and return the previous setting to the client application. Please refer to 4.6.1 Option ID codes for details on options and their effects. 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

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

4.6.1 Option ID codes

3.8.4.TSI_MISC_GetErrorDescription

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_MISC_GetErrorDescription
(
     TSI_RESULT ErrorCode,
     char *ErrorString,
     unsigned int StringMaxLen
);
```

Synopsis

Retrieves a human readable error message matching the given ErrorCode.

Parameters

ErrorCode

Indicates the error code to identify.

ErrorString

Pointer to an array of characters that will receive a human readable description of the error code. The resulting string may contain newlines. The string is guaranteed to be NULL terminated. If the string buffer is not large enough to store the complete description string, the string is truncated.

StringMaxLen

Maximum length of the string in characters. Recommended size for error message strings is 128 chars or more.

Result

If the function succeeds, the return value is the number of characters required for the complete error description string. If the return value is EQUAL or HIGHER than StringMaxLen parameter's value, it means that the string was truncated.

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

See Also

4.5 Error codes

3.9.Status log functions

3.9.1.TSI_STLOG_GetMessageCount

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_STLOG_GetMessageCount();
```

Synopsis

Retrieves the number of queued status log message.

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

3.9.2 TSI_STLOG_Clear, 3.9.3 TSI_STLOG_GetMessageData

3.9.2.TSI_STLOG_Clear

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_STLOG_Clear();
```

Synopsis

Clear the status log buffer.

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

-

3.9.3.TSI_STLOG_GetMessageData

ClientVersion 1, and higher

```
TSI_RESULT __stdcall TSI_STLOG_GetMessageData
(
     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

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

3.9.1 TSI_STLOG_GetMessageCount

3.9.4.TSI_STLOG_WaitMessage

ClientVersion 5, and higher

```
TSI_RESULT __stdcall TSI_STLOG_WaitMessage
(
     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

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

3.9.1 TSI_STLOG_GetMessageCount and 3.9.3 TSI_STLOG_GetMessageData

3.10.Report generator functions

3.10.1.TSI_REP_BeginLogRecord

ClientVersion 5, and higher

```
TSI_RESULT __stdcall TSI_REP_BeginLogRecord
(
     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

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

3.10.2 TSI_REP_EndLogRecord

3.10.2.TSI_REP_EndLogRecord

ClientVersion 5, and higher

```
TSI_RESULT __stdcall TSI_REP_EndLogRecord();
```

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.

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

3.10.1 TSI_REP_BeginLogRecord

4. TYPES, DEFAULTS AND DEFINITIONS

This chapter describes type definitions, default values and defined constants.

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;
```

4.2.Defaults

API defaults are listed here.

4.2.1.Default capture device

The default capture device has Device ID of zero. The devices ID's are assigned by sorting the supported devices into a list towards increasing serial number. Therefore, the default device is the one with lowest serial number of all supported devices.

4.2.2.Default input

Default input is defined per device. For UFG-06 the default input is the HDMI-1 input.

4.3.Tests

All tests, and their associated configurations and requirements are listed here.

4.3.1.Compare video frame sequence with a single reference

```
ClientVersion 1, and later
```

```
#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

TSI_PARAGRP_REFERENCE_1

The test always uses reference frame 1.

No default reference frame exists: the reference frame configuration must be set somehow before the test can be executed – This can be done by capturing a reference frame or by loading it from disk, or through other means: Please refer to Reference frame configuration items for exact details in which configuration items to set.

TSI_TEST_LENGTH

Number of frames to capture and compare to reference. Default setting is 60.

Important: *Capturing a high-resolution frames into system RAM will consume a considerable amount of memory. If a memory error is encountered when trying to start this test, try reducing this value.*

TSI_LIM_FRAME_MISMATCHES

If the number of “bad” frames exceeds this number during the comparison stage, the test outcome will be “failed”. Default setting is 0.

TSI_LIM_PIXEL_MISMATCHES

If the number of failed pixels (per frame) exceeds this number when comparing a single frame to reference, the compared frame is considered “bad”. Default setting is 0.

TSI_PIXEL_TOLERANCE

This test allows deviation between captured and reference frame pixels. The deviation is calculated for each color-channel of each pixel. If the deviation on any channel exceeds this value, the pixel is considered “bad”. Default setting is 0.

(Continued...)

(...Continued)

Additional/optional configuration items

In addition to just giving pass/fail result, the test can also be programmed to save failed frames on disk automatically. To enable auto-saving the following configuration items must be programmed:

TSI_MAX_AUTO_SAVE_FAILED
: Maximum number of failed frames saved into target folder per test. To enable auto-saving, this configuration item must be set to non-zero value. Default setting is 0.

  Important: *Auto-saving will never overwrite files, therefore it is necessary for the client application to watch the size of the directory so that the application will not fill out the entire hard-disk with failed frame data. Also, folders that have thousands of files will slow down saving new files.*

TSI_FAILED_FRAME_TARGET_FOLDER
: A character string that identifies the target folder into which the failed frames are saved into. The file-names are “Failed_#.bmp”, with the '#'-char replaced with a serial number for the frame. No default.

  Important: *If this feature is enabled, this setting should be assigned to a known location. If nothing is assigned, there may be unexpected behavior.*

If an application requires access to the failed frames, but does not need/want them to be automatically saved on disk, the test can also be programmed to provide access to the failed frames. The following configuration items are used to enable and access the failed frames:

TSI_MAX_EXPORT_FAILED
: Maximum number of failed frames exported from a single test run. To enable exports, this setting must be set to a non-zero value. Default setting is 0.

  Important: *The frames are available until any other test is started. Starting a test will release the memory allocations.*

TSI_EXPORTED
: **READ ONLY**. Number of frames currently exported.

TSI_EXPORT_ACCESS_INDEX
: **WRITE ONLY**. Which frame of the currently exported frames to access. Valid range is from zero (0) to value from *TSI_EXPORTED* minus one.

TSI_EXPORT_WIDTH
: **READ ONLY**. Width of the frame, in elements. The frame being accessed is indicated by the *TSI_EXPORT_ACCESS_INDEX* configuration item.

TSI_EXPORT_HEIGHT
: **READ ONLY**. Height of the frame, in elements. The frame being accessed is indicated by the *TSI_EXPORT_ACCESS_INDEX* configuration item.

TSI_EXPORT_ELEMENT_SIZE
: **READ ONLY**. The size of a single element, in bytes. The frame being accessed is indicated by the *TSI_EXPORT_ACCESS_INDEX* configuration item.

(Continued...)

(...Continued)

TSI_EXPORT_PIXELS_PER_ELEMENT

READ ONLY. The width of a single element, in real pixels. The frame being accessed is indicated by the *TSI_EXPORT_ACCESS_INDEX* configuration item.

TSI_EXPORT_LINES_PER_ELEMENT

READ ONLY. The height of a single element, in real pixels. The frame being accessed is indicated by the *TSI_EXPORT_ACCESS_INDEX* configuration item.

TSI_EXPORT_COLOR_DEPTH

READ ONLY. Color depth as bits per color channel. The frame being accessed is indicated by the *TSI_EXPORT_ACCESS_INDEX* configuration item.

TSI_EXPORT_PIXEL_FORMAT

READ ONLY. Identifies the color encoding used within the element. The frame being accessed is indicated by the *TSI_EXPORT_ACCESS_INDEX* configuration item.

TSI_EXPORT_FRAME_DATA

READ ONLY. Contains the RAW frame data as described by the previous configuration items. The frame being accessed is indicated by the *TSI_EXPORT_ACCESS_INDEX* configuration item.

The application can also read exact failed pixel counts per frame. This data is gathered automatically, and is available until the next video test run is started.

TSI_R_VIDEO_TEST_RAW_RESULTS_DATA

READ ONLY. Provides access to RAW results generated by the video test. This configuration item has variable size, so please determine it's size before attempting to read the results into a buffer. The RAW results block is a list of unsigned integers, see the table below for description of the resulting list:

Index	Description
0	Total number of failed sub-pixels found on red color channel for frame 1
1	Total number of failed sub-pixels found on green color channel for frame 1
2	Total number of failed sub-pixels found on blue color channel for frame 1
3	Total failed pixels for frame 1
4	Total number of failed sub-pixels found on red color channel for frame 2
5	Total number of failed sub-pixels found on green color channel for frame 2
6	Total number of failed sub-pixels found on blue color channel for frame 2
7	Total failed pixels for frame 2
8 … n	… Additional data for following frames ...

(Continued...)

(...Continued)

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
```

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”.

See Also

4.4 Configuration item definitions

4.3.2.Validate audio signal frequency and glitch-free audio reproduction

```
ClientVersion 4, and later
```

```
#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

TSI_EXPECTED_SAMPLE_RATE
: The expected samples per second of the digital audio stream. If the actual sample rate does not match this value, the test can't be executed. Default setting is 44100.

TSI_EXPECTED_AUDIO_FREQUENCY
: The frequency that expected to have the highest power in the spectrum, in Hz. Default setting 1000.

TSI_AUDIO_FREQUENCY_TOLERANCE
: The allowed deviation between the measured highest-power and the expected highest power, in Hz. Default setting is 1.

TSI_AUDIO_GLITCH_DETECT_TRESHOLD
: 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.

  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.*

(Continued...)

(...Continued)

TSI_AUDIO_GLITCHES_ALLOWED
Number of detected glitches allowed per test.

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.*

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.

See Also

4.4 Configuration item definitions

4.3.3.HDMI Electrical tests, HDMI Source power line test

```
ClientVersion 7, and higher
```

```
#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

TSI_HDMI_RX_TIMEOUT
: Timeout for the electrical test in milliseconds. Default setting is 5000 ms.

TSI_HDMI_RX_POWER_LOW_LIMIT
: Lower voltage limit for the power line test, in millivolts. Default setting is 4700 mV.

TSI_HDMI_RX_POWER_HIGH_LIMIT
: Higher voltage limit for the power line test, in millivolts. Default setting is 5300 mV.

See Also

4.4 Configuration item definitions

4.3.4.HDMI Electrical tests, HDMI Source HPD line test

ClientVersion 7, and higher

```
#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.

Configuration items

TSI_HDMI_RX_TIMEOUT
: Timeout for the electrical test in milliseconds. Default setting 5000 ms.

TSI_HDMI_RX_HPD_ZERO_LOW_LIMIT
: HPD Logical zero voltage window – lower allowed voltage boundary, in millivolts. Default setting is 0 mV.

TSI_HDMI_RX_HPD_ZERO_HIGH_LIMIT
: HPD Logical zero voltage window – higher allowed voltage boundary, in millivolts. Default setting is 400 mV.

TSI_HDMI_RX_HPD_ONE_LOW_LIMIT
: HPD Logical one voltage window – Lower allowed voltage boundary, in millivolts. Default setting is 2400 mV.

TSI_HDMI_RX_HPD_ONE_HIGH_LIMIT
: HPD Logical one voltage window – higher allowed voltage boundary, in millivolts. Default setting is 5300 mV.

See Also

4.4 Configuration item definitions

4.3.5.HDMI Electrical tests, HDMI DDC and CEC lines test

```
ClientVersion 7, and higher
```

```
#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

TSI_HDMI_RX_TIMEOUT
: Timeout for the electrical test in milliseconds. Default setting 5000 ms.

TSI_HDMI_RX_DDC_LOW_LIMIT
: DDC line voltage window – Lower allowed voltage boundary, in millivolts. Default is 4500 mV.

TSI_HDMI_RX_DDC_HIGH_LIMIT
: DDC line voltage window – Higher allowed voltage boundary, in millivolts. Default is 5500 mV.

TSI_HDMI_RX_CEC_ZERO_LOW_LIMIT
: CEC logical zero voltage window – Lower allowed voltage boundary, in millivolts. Default is 0 mV.

TSI_HDMI_RX_CEC_ZERO_HIGH_LIMIT
: CEC logical zero voltage window – Higher allowed voltage boundary, in millivolts. Default is 600 mV.

TSI_HDMI_RX_CEC_ONE_LOW_LIMIT
: CEC logical one voltage window – Lower allowed voltage boundary, in millivolts. Default is 2500 mV.

TSI_HDMI_RX_CEC_ONE_HIGH_LIMIT
: CEC logical one voltage window – Higher allowed voltage boundary, in millicolts. Default is 3600 mV.

See Also

4.4 Configuration item definitions

4.3.6.HDMI Electrical tests, HDMI TMDS line test

ClientVersion 7, and higher

```
#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

TSI_HDMI_RX_TIMEOUT
: Timeout for the electrical test in milliseconds. Default setting 5000 ms.

TSI_HDMI_RX_LINK_LOW_LIMIT
: HDMI TMDS link line voltage window – Lower allowed voltage boundary, in millivolts. Default is 2900 mV.

TSI_HDMI_RX_LINK_HIGH_LIMIT
: HDMI TMDS link line voltage window – Higher allowed voltage boundary, in millivolts. Default is 3100 mV.

See Also

4.4 Configuration item definitions

4.3.7.HDMI 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

TSI_HDMI_RX_CEC_TIMEOUT
: HDMI CEC test timeout in milliseconds. Default is 5000 ms.

TSI_HDMI_RX_CEC_LOCAL_PHY_ADDR
: The local CEC physical address used for generating downstream physical addresses. Please, see HDMI CEC specification for details. Default is 4.0.0.0 (0x4000)

See Also

4.4 Configuration item definitions

4.3.8.DP Electrical tests, DP Main Link lines test

ClientVersion 7, and higher

```
#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

TSI_DP_RX_TEST_TIMEOUT
: Timeout for the electrical test in milliseconds. Default value is 5000 ms.

TSI_DP_RX_LINKS_LOW_VOLTAGE
: DP link power window, lower voltage boundary in millivolts. Default is 2600 mV.

TSI_DP_RX_LINKS_HI_VOLTAGE
: DP link power window, higher voltage boundary in millivolts. Default is 4000 mV.

TSI_DP_RX_MAX_DUT_LANE_COUNT
: Max. number of lanes supported by the DUT. Default is 4.

TSI_DP_RX_MAX_DUT_LINK_RATE
: Max. lane frequency as multiple of 0.27Gbps. Default is 20 (5.4Gbps).

TSI_DP_RX_DUT_CAPABILITIES
: DUT Capability flags. Currently no flags defined. Default is 0.

TSI_DP_RX_DUT_TEST_AUTOMATION_FLAGS
: DUT Test automation support flags. Default is 0.

See Also

4.4 Configuration item definitions

4.3.9.DP Electrical tests, DP AUX Lines test

ClientVersion 7, and higher

```
#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

TSI_DP_RX_TEST_TIMEOUT
: Timeout for the electrical test in milliseconds. Default value is 5000 ms.

TSI_DP_RX_AUX_P_IDLE_LOW_VOLTAGE
: DP AUX P-Line idle state voltage window, lower boundary voltage in millivolts. Default 20 mV.

TSI_DP_RX_AUX_P_IDLE_HI_VOLTAGE
: DP AUX P-Line idle state voltage window, higher boundary voltage in millivolts. Default 500 mV.

TSI_DP_RX_AUX_N_IDLE_LOW_VOLTAGE
: DP AUX N-Line idle state voltage window, lower boundary voltage in millivolts. Default 2600 mV.

TSI_DP_RX_AUX_N_IDLE_HI_VOLTAGE
: DP AUX N-Line idle state voltage window, higher boundary voltage in millivolts. Default 3600 mV.

TSI_DP_RX_AUX_P_TRIG_VOLTAGE
: DP AUX P-Line trigger voltage level in millivolts. Default is 150 mV.

TSI_DP_RX_AUX_N_TRIG_VOLTAGE
: DP AUX N-Line trigger voltage level in millivolts. Default is 200 mV.

TSI_DP_RX_AUX_SIGNAL_CAPT_TIMEOUT
: Period of time TE waits for AUX transactions starting from end of HPD pulse. Default is 200 ms.

TSI_DP_RX_AUX_SIGNAL_CAPT_TRIES
: If AUX transactions are not detected within the timeout period TW will try again. Default is 5.

See Also

4.4 Configuration item definitions

4.3.10.DP Electrical tests, HPD line test

ClientVersion 7, and later

```
#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

TSI_DP_RX_TEST_TIMEOUT
: Timeout for the electrical test in milliseconds. Default value is 5000 ms.

TSI_DP_RX_HPD_ZERO_LOW_VOLTAGE
: HPD line logical zero voltage window, lower boundary in millivolts. Default is -100 mV.

TSI_DP_RX_HPD_ZERO_HI_VOLTAGE
: HPD line logical zero voltage window, higher boundary in millivolts. Default is 800 mV.

TSI_DP_RX_HPD_ONE_LOW_VOLTAGE
: HPD line logical one voltage window, lower boundary in millivolts. Default is 800 mV.

TSI_DP_RX_HPD_ONE_HI_VOLTAGE
: HPD line logical one voltage window, higher boundary in millivolts. Default is 5500 mV.

See Also

4.4 Configuration item definitions

4.3.11.CRC based single frame video stability test

ClientVersion 8, and higher

```
#define TSI_TEST_VIDEO_CRC_SINGLE_REF          0x00060000
```

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.

Configuration items

TSI_CRC_TIMEOUT
Indicates test timeout in milliseconds. Default value is 1000ms

TSI_CRC_FRAMES_TO_TEST
Indicates number of frames to be tested. Default value is 20.

Important: *Make sure the TSI_CRC_TIMEOUT is long enough to allow this many frames to be received.*

TSI_CRC_LIM_FRAME_MISMATCHES
Number of frames that are allowed to have mismatching CRC with the reference frame(s). Default value is 0.

TSI_CRC_REF_WIDTH
Width of the reference frame used, in pixels. Default value is 1920.

TSI_CRC_REF_HEIGHT
Height of the reference frame used, in pixels. Default value is 1080.

TSI_CRC_REF_COLORDEPTH
Bit depth of the reference frame used, in bits per pixel. Default value is 24.

TSI_CRC_REFERENCE_CRC_VALUES
A block of CRC reference value sets. Each CRC reference value set is a block of 3 16-bit values, with Red/Cr CRC at the lowest address, and Blue/Cb CRC at the highest address.

See Also

4.4 Configuration item definitions

4.3.12.CRC based single frame video stability test

ClientVersion 8, and higher

```
#define TSI_TEST_CRC_VIDEO_STABILITY          0x00060001
```

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.

Configuration items

TSI_CRC_TIMEOUT
: Indicates test timeout in milliseconds. Default value is 1000ms

TSI_CRC_FRAMES_TO_TEST
: Indicates number of frames to be tested. Default value is 20.

  Important: *If the value is zero, the test will run until time out, but still passes if no mismatches were detected.*

  Important: *Make sure the TSI_CRC_TIMEOUT is long enough to allow this many frames to be received.*

See Also

4.4 Configuration item definitions

4.3.13.CRC based sequence of frames reference video test

ClientVersion 8, and higher

```
#define TSI_TEST_CRC_VIDEO_SEQUENCE            0x00060002
```

Synopsis

The Source DUT should be sending a repeating video sequence to the TE, without the sequence containing no identical frames within the loop.

The test will first synchronize with the provided CRC sequence by finding the by finding video frame with the CRC matching 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. 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.

TSI_CRC_TIMEOUT
: Indicates test timeout in milliseconds. Default value is 1000ms

TSI_CRC_FRAMES_TO_TEST
: Indicates number of frames to be tested. Default value is 20.

> ***Important:*** *Make sure the TSI_CRC_TIMEOUT is long enough to allow this many frames to be received.*

TSI_CRC_REF_WIDTH
: Width of the reference frame used, in pixels. Default value is 1920.

TSI_CRC_REF_HEIGHT
: Height of the reference frame used, in pixels. Default value is 1080.

TSI_CRC_REF_COLORDEPTH
: Bit depth of the reference frame used, in bits per pixel. Default value is 24.

TSI_CRC_REFERENCE_CRC_VALUES
: A block of CRC reference value sets. Each CRC reference value set is a block of 3 16-bit values, with Red/Cr CRC at the lowest address, and Blue/Cb CRC at the highest address.

See Also

4.4 Configuration item definitions

4.4.Configuration item definitions

The following chapters list the available configuration items.

4.4.1.Reference frame configuration items

The table below defines the configuration items that form a reference frame. The configuration item enumeration functions may use TSI_PARAGRP_REFERENCE_1 to refer to all of the configurations items defined here.

These configuration items are available on all devices that support video capture. These configuration items have no default values.

Define	Config ID	Default	Description
TSI_REF1_WIDTH	0x10	N/A	Frame width in number of elements. A single element can contain one or more pixels depending on frame format.
TSI_REF1_HEIGHT	0x11	N/A	Frame height in number of elements. A single element can contain one or more lines depending on frame format.
TSI_REF1_ELEMENT_SIZE	0x12	N/A	Size of a single element in bytes.
TSI_REF1_PIXELS_PER_ELEMENT	0x13	N/A	Width of a single element in pixels. Use the lowest possible value.
TSI_REF1_LINES_PER_ELEMENT	0x14	N/A	Height of a single element in pixels. Use the lowest possible value.
TSI_REF1_COLOR_DEPTH	0x15	N/A	Number of bits per color channel.
TSI_REF1_PIXEL_FORMAT	0x16	N/A	Pixel format ID. 0 = RGB 4:4:4
TSI_REF1_FRAME_DATA	0x17	N/A	Pointer to image data. Image buffer size must be WIDTH * HEIGHT * ELEMENT_SIZE bytes. ***Important:*** *Assign the other TSI_REF1_xxx configuration items before assigning the buffer.*

4.4.2.Configuration items for video format read

The configuration items in the table below provide a method to read information about the video stream currently being received by the device.

Define	Config ID	Default	Description
TSI_R_INPUT_WIDTH	0x200	N/A	***READ ONLY.*** Video frame width as number of elements.
TSI_R_INPUT_HEIGHT	0x201	N/A	***READ ONLY.*** Video frame height as number of elements.
TSI_R_INPUT_FREQ	0x202	N/A	***READ ONLY.*** Video input frame rate as multiple of 0.1 Hz frequency.
TSI_R_INPUT_ELEMENTS_SIZE	0x203	N/A	***READ ONLY.*** Size of a single frame element in bytes.
TSI_R_INPUT_PIXELS_PER_ELEMENT	0x204	N/A	***READ ONLY.*** Width of single element as pixels.
TSI_R_INPUT_LINES_PER_ELEMENT	0x205	N/A	***READ ONLY.*** Height of single element as lines.
TSI_R_INPUT_COLOR_DEPTH	0x206	N/A	***READ ONLY.*** Number of bits per color channel.
TSI_R_INPUT_PIXEL_FORMAT	0x207	N/A	***READ ONLY.*** Pixel format ID code. 0x000 = RGB 8:8:8 0x001 = RGB 16:16:16 0x100 = YCbCr 8:8:8 0x101 = YCbCr 16:16:16
TSI_R_INPUT_INTERLACE	0x208	N/A	***READ ONLY.*** Indicates current timing interlace status 0 = Progressive 1 = Interlaced

4.4.3.Configuration items for audio format read

The configuration items in the table below provide a method to read information about audio stream currently being received by the device.

Define	Config ID	Default	Description
TSI_R_AUDIO_CHANNELS	0x220	N/A	***READ ONLY.*** Active audio channels (1 to 8)
TSI_R_AUDIO_SAMPLE_RATE	0x221	N/A	***READ ONLY.*** Audio sampling rate in Hz.
TSI_R_AUDIO_SAMPLE_SIZE	0x222	N/A	***READ ONLY.*** PCM Sample size, in bits.

4.4.4.Configuration items for video capture

The configuration items in the table below provide a method to set capture operation related settings for devices that support them. Please see device specific documentation to see if these configuration items are supported on a particular device.

Define	Config ID	Default	Description
TSI_CAPTURE_COLOR_DEPTH	0x230	N/A	Selects capture color-depth used when capturing from a device. -1 = Auto select (Match input if possible) 0 = 6 bits per color channel 1 = 8 bits per color channel 2 = 10 bits per color channel 3 = 12 bits per color channel
TSI_CAPTURE_COLOR_SPACE	0x231	N/A	Selects capture color-space used when capturing from a device. -1 = Auto select (Match input if possible) 0 = RGB

4.4.5.Configuration items for Audio capture

The configuration items in the table below provide a method to set audio capture related settings for devices that support them. Please see device specific documentation to see if the configuration items are supported on a particular device.

Define	Config ID	Default	Description
TSI_CAPTURE_AUDIO_MASK	0x238	3	This configuration item is supported only on devices that cannot automatically determine active audio channels, and must rely on client application defining the active channels instead. The value is 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 active audio channels 0 and 1, or stereo.

4.4.6.Configuration items related to V-by-One inputs

The configuration items are used to configure V-by-One inputs.

Define	Config ID	Default	Description
TSI_VX1_SIGNAL_COLOR_DEPTH	0x240	N/A	Defines V-by-One 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
TSI_VX1_SIGNAL_CHANNELS_PER_UNIT	0x241	N/A	Defines number of V-by-One channels to capture per capture unit: 1, 2, 4 or 8. Actual number of channels is this setting multiplied by TSI_R_UNITS_PRESENT value. See Error: Reference source not found Error: Reference source not found for details. ***Important:*** *Setting this value also sets TSI_VX1_SECTION_COUNT configuration item value to it's default value (1).*
TSI_VX1_SIGNAL_SYNC_MODE	0x242	0	0 = Data Enable, 1 = HSync + VSync.
TSI_VX1_HTPDN_CONTROL	0x243	0	0 = Normal operation 1 = Force Low 2 = Force High
TSI_VX1_LOCKN_CONTROL	0x244	0	0 = Normal operation 1 = Force Low 2 = Force High 3 = Force Low after TSI_VX1_LOCKN_DELAY
TSI_VX1_LOCKN_DELAY	0x245	41900	Delay, in µs. Used only TSI_VX1_LOCKN_CONTROL is set to 3; Otherwise ignored.
TSI_VX1_VIDEO_VALID_DELAY	0x246	41900	Delay, in µs.
TSI_VX1_FRAME_COMBINE_METHOD	0x247	0	Identifies how the frames received from multiple V-by-One units are merged together. 0 = Default (One from each lane in sequence)
TSI_VX1_SECTION_COUNT	0x248	1	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 configuration items TSI_R_UNITS_PRESENT and TSI_VX1_SIGNAL_CHANNELS_PER_UNIT ***Important:*** *Set this configuration item after setting the item TSI_VX1_SIGNAL_CHANNELS_PER_UNIT. Setting TSI_VX1_SIGNAL_CHANNELS_PER_UNIT also sets number of sections to 1.*

4.4.7.Configuration items for Test ID 2

The configuration items below are used with test “Compare video frame sequence with a single reference frame”.

Define	Config ID	Default	Description
TSI_TEST_LENGTH	0x1000	60	Number of consequent frames that are checked.
TSI_LIM_FRAME_MISMATCHES	0x1001	0	Number of failed frames allowed during a test.
TSI_LIM_PIXEL_MISMATCHES	0x1002	0	Number of failed pixels per frame allowed.
TSI_PIXEL_TOLERANCE	0x1003	0	Absolute difference between a reference image and the captured image allowed per pixel.
TSI_MAX_AUTO_SAVE_FAILED	0x1080	0	Max. number of failed frames saved.
TSI_FAILED_FRAME_TARGET_FOLDER	0x1081	N/A	String: Auto save frames target folder.
TSI_MAX_EXPORT_FAILED	0x1082	0	Max. number of failed frames to export per test.
TSI_R_VIDEO_TEST_RAW_RESULTS_DATA			***READ ONLY.*** When read, returns a count of unsigned integer values that conveys the measured pixel error counts from previous run of test ID 2. Please refer to 4.3.1 Compare video frame sequence with a single reference for more details.
TSI_EXPORTED	0x108e	N/A	***READ ONLY.*** Number of frames currently exported.
TSI_EXPORT_ACCESS_INDEX	0x108f	N/A	***WRITE ONLY.*** Number of exported frame accessed.
TSI_EXPORT_WIDTH	0x1090	N/A	***READ ONLY.*** Frame width as number of elements.
TSI_EXPORT_HEIGHT	0x1091	N/A	***READ ONLY.*** Frame height as number of elements.
TSI_EXPORT_ELEMENT_SIZE	0x1092	N/A	***READ ONLY.*** Size of a single frame element in bytes.
TSI_EXPORT_PIXELS_PER_ELEMENT	0x1093	N/A	***READ ONLY.*** Width of single element as pixels.
TSI_EXPORT_LINES_PER_ELEMENT	0x1094	N/A	***READ ONLY.*** Height of single element as lines.
TSI_EXPORT_COLOR_DEPTH	0x1095	N/A	***READ ONLY.*** Number of bits per color channel.
TSI_EXPORT_PIXEL_FORMAT	0x1096	N/A	***READ ONLY.*** Pixel format ID code. 0 = RGB 4:4:4
TSI_EXPORT_FRAME_DATA	0x1097	N/A	***READ ONLY.*** Pointer to image data. Image buffer size must be WIDTH * HEIGHT * ELEMENT_SIZE bytes.

4.4.8.Configuration items for Test ID 3

The configuration items below are used with test “Validate audio signal frequency and glitch-free audio reproduction”.

Define	Config ID	Default	Description
TSI_EXPECTED_SAMPLE_RATE	0x2020	44100	Audio sampling rate used by source. In Hz.
TSI_EXPECTED_AUDIO_FREQUENCY	0x2021	1000	Expected sine-wave audio frequency carried. In Hz.
TSI_AUDIO_FREQUENCY_TOLERANCE	0x2022	1	Allowed deviation for expected sine-wave audio frequency. In Hz.
TSI_AUDIO_GLITCH_DETECT_TRESHOLD	0x2023	5.0 (327680)	Threshold value used to detect audible clicks caused by dropped or duplicated samples. ***FIXED POINT:*** top 16-bits are the integer part, and low 16 bits are fraction.
TSI_AUDIO_GLITCHES_ALLOWED	0x2024	0	Number of audio glitches allowed per test.

4.4.9.Configuration items for HDMI Receiver Electrical tests

The configuration items below are used with HDMI Electrical tests. (UCD-3xx family devices that have electrical test hardware installed)

Define	Config ID	Default	Description
TSI_HDMI_RX_TIMEOUT	0x10200	5000	Electrical test timeout in milliseconds.
TSI_HDMI_RX_POWER_LOW_LIMIT	0x10201	4700	HDMI power-supply low limit, in millivolts (mV)
TSI_HDMI_RX_POWER_HIGH_LIMIT	0x10202	5300	HDMI power-supply high limit, in millivolts (mV)
TSI_HDMI_RX_LINK_LOW_LIMIT	0x10203	2900	HDMI Link voltage low limit, in millivolts (mV)
TSI_HDMI_RX_LINK_HIGH_LIMIT	0x10204	3100	HDMI Link voltage high limit, in millivolts (mV)
TSI_HDMI_RX_HPD_ZERO_LOW_LIMIT	0x10205	0	HDMI HPD logical zero voltage level low limit, in millivolts (mV)
TSI_HDMI_RX_HPD_ZERO_HIGH_LIMIT	0x10206	400	HDMI HPD logical zero voltage level high limit, in millivolts (mV)
TSI_HDMI_RX_HPD_ONE_LOW_LIMIT	0x10207	2400	HDMI HPD logical one voltage level low limit, in millivolts (mV)
TSI_HDMI_RX_HPD_ONE_HIGHT_LIMIT	0x10208	5300	HDMI HPD logical one voltage level high limit, in millivolts (mV)
TSI_HDMI_RX_DDC_LOW_LIMIT	0x10209	4500	HDMI DDC voltage level low limit, in millivolts (mV)
TSI_HDMI_RX_DDC_HIGH_LIMIT	0x1020a	5500	HDMI DDC voltage level high limit, in millivolts (mV)
TSI_HDMI_RX_CEC_ZERO_LOW_LIMIT	0x1020b	0	HDMI CEC Logical zero low voltage limit, in millivolts (mV)
TSI_HDMI_RX_CEC_ZERO_HIGH_LIMIT	0x1020c	600	HDMI CEC logical zero hight voltage limit, in millivolts (mV)
TSI_HDMI_RX_CEC_ONE_LOW_LIMIT	0x1020d	2500	HDMI CEC logical one low voltage limit, in millivolts (mV)
TSI_HDMI_RX_CEC_ONE_HIGH_LIMIT	0x1020e	3600	HDMI CEC logical one high voltage limit, in millivolts (mV)

4.4.10.Configuration items for DP Receiver Electrical tests

The configuration items below are used with DP Electrical tests. (UCD-3xx family devices that have electrical test hardware installed)

Define	Config ID	Default	Description
TSI_DP_RX_TEST_TIMEOUT	0x10100	5000	Electrical test timeout in milliseconds
TSI_DP_RX_LINKS_LOW_VOLTAGE	0x10101	2600	DP data links low voltage in millivolts (mV)
TSI_DP_RX_LINKS_HI_VOLTAGE	0x10102	4000	DP data links hi voltage in millivolts (mV)
TSI_DP_RX_HPD_ZERO_LOW_VOLTAGE	0x10103	-100	DP HPD line logical zero voltage level low limit, in millivolts (mV)
TSI_DP_RX_HPD_ZERO_HI_VOLTAGE	0x10104	800	DP HPD line logical zero voltage level hi limit, in millivolts (mV)
TSI_DP_RX_HPD_ONE_LOW_VOLTAGE	0x10105	800	DP HPD line logical one voltage level low limit, in millivolts (mV)
TSI_DP_RX_HPD_ONE_HI_VOLTAGE	0x10106	5500	DP HDP line logical one voltage level hi limit, in millivolts (mV)
TSI_DP_RX_AUX_P_IDLE_LOW_VOLTAGE	0x10107	20	DP AUX P-Line idle state low voltage limit, in millivolts (mV)
TSI_DP_RX_AUX_P_IDLE_HI_VOLTAGE	0x10108	500	DP AUX P-Line idle state hi voltage limit, in millivolts (mV)
TSI_DP_RX_AUX_N_IDLE_LOW_VOLTAGE	0x10109	2600	DP AUX N-Line idle state low voltage limit, in millivolts (mV)
TSI_DP_RX_AUX_N_IDLE_HI_VOLTAGE	0x1010a	3600	DP AUX N-Line idel state hi voltage limit, in millivolts (mV)
TSI_DP_RX_AUX_P_TRIG_VOLTAGE	0x1010b	150	DP AUX P-Line trigger voltage level in millivolts (mV)
TSI_DP_RX_AUX_N_TRIG_VOLTAGE	0x1010c	200	DP AUX N-Line trigger voltage level, in millivolts (mV)
TSI_DP_RX_AUX_SIGNAL_CAPT_TIMEOUT	0x1010d	200	DP AUX Signal capture timeout, in milliseconds (ms)
TSI_DP_RX_AUX_SIGNAL_CAPT_TRIES	0x1010e	5	DP AUX Signal capture retries.
TSI_DP_RX_MAX_DUT_LANE_COUNT	0x1010f	4	DP DUT Max. lanes.
TSI_DP_RX_MAX_DUT_LINK_RATE	0x10110	20	DP DUT Max. lane frequency as multiplier of 0.27Gbps
TSI_DP_RX_DUT_CAPABILITIES	0x10111	0	DUT Capability flags (flags are TBD)
TSI_DP_RX_DUT_TEST_AUTOMATION_FLAGS	0x10112	0	DUT Test automation flags. Bit #0 = Test link training Bit #1 = Test video pattern Bit #2 = Test EDID read

4.4.11.Configuration items for reading info frames

The configuration items that can be used to read info-frames from a device that support reading them. (UCD-3xx family). Please refer to appropriate standards to decode the produced data.

Define	Config ID	Default	Description
TSI_R_HDMI_INFOFRAME_RANGE_START	0x11000	N/A	***READ ONLY.*** First configuration item that can be used to access HDMI info-frames. To access, for example AVI info frame, you can use the direct define, or add the AVI info frame's type to this configuration item's ID value. In future releases this style can be used to access infoframes that do not have a define specified in TSI.
TSI_R_HDMI_INFOFRAME_RANGE_END	0x110ff	N/A	***READ ONLY.*** Last configuration item that can be used to access HDMI info-frames.
TSI_R_HDMI_INFOFRAME_NULL	0x11000	N/A	***READ ONLY.*** NULL infoframe.
TSI_R_HDMI_INFOFRAME_ACR	0x11001	N/A	***READ ONLY.*** Audio Clock Regeneration.
TSI_R_HDMI_INFOFRAME_ASP	0x11002	N/A	***READ ONLY.*** Audio Sample Packet
TSI_R_HDMI_INFOFRAME_GCP	0x11003	N/A	***READ ONLY.*** General Control Packet
TSI_R_HDMI_INFOFRAME_ACP	0x11004	N/A	***READ ONLY.*** Audio Content Protection packet
TSI_R_HDMI_INFOFRAME_ISRC1	0x11005	N/A	***READ ONLY.*** International Standard Recording Code
TSI_R_HDMI_INFOFRAME_ISRC2	0x11006	N/A	***READ ONLY.*** International Standard Recording Code
TSI_R_HDMI_INFOFRAME_OBA	0x11007	N/A	***READ ONLY.*** One Bit Audio sample packet
TSI_R_HDMI_INFOFRAME_DTS	0x11008	N/A	***READ ONLY.*** DTS audio packet
TSI_R_HDMI_INFOFRAME_HBR	0x11009	N/A	***READ ONLY.*** High BitRate audio stream packet
TSI_R_HDMI_INFOFRAME_GMP	0x1100a	N/A	***READ ONLY.*** Gamut Metadata Packet
TSI_R_HDMI_INFOFRAME_VSI	0x11081	N/A	***READ ONLY.*** Vendor Specific Infoframe
TSI_R_HDMI_INFOFRAME_AVI	0x11082	N/A	***READ ONLY.*** Auxiliary Video Information infoframe
TSI_R_HDMI_INFOFRAME_SPD	0x11083	N/A	***READ ONLY.*** Source Product Descriptor infoframe
TSI_R_HDMI_INFOFRAME_AIF	0x11084	N/A	***READ ONLY.*** Audio Infoframe
TSI_R_HDMI_INFOFRAME_MPEG	0x11085	N/A	***READ ONLY.*** MPEG Source infoframe
TSI_R_HDMI_INFOFRAME_3D_ASP	0x1100b	N/A	***READ ONLY.*** 3D Audio Sample Packet
TSI_R_HDMI_INFOFRAME_3D_OBA	0x1100c	N/A	***READ ONLY.*** 3D One Bit Audio sample packet
TSI_R_HDMI_INFOFRAME_AMP	0x1100d	N/A	***READ ONLY.*** Audio Metadata Packet
TSI_R_HDMI_INFOFRAME_MST_ASP	0x1100e	N/A	***READ ONLY.*** Multi-stream Audio Sample Packet
TSI_R_HDMI_INFOFRAME_MST_OBA	0x1100f	N/A	***READ ONLY.*** Multi-stream One Bit Audio sample packet
TSI_R_HDMI_INFOFRAME_DRM	0x11087	N/A	***READ ONLY.*** Dynamic Range and Mastering infoframe

(Continued…)

(...Continued)

<table>
<tr><th>Define</th><th>Config ID</th><th>Default</th><th colspan="2">Description</th></tr>
<tr><td rowspan="28">TSI_R_HDMI_INFOFRAME_UPDATE_FLAGS</td><td rowspan="28">0x11100</td><td rowspan="28">N/A</td><td colspan="2">READ ONLY; CLEAR ON READ. 256-bit flags field stored as a little-endian 256-bit word. Bits set to one indicate new data is available since previous read of this bit-field. Cleared bits indicate data is not updated since last read of this bit-field.</td></tr>
<tr><td>Bit #</td><td>Description</td></tr>
<tr><td>0</td><td>NULL (May not contain non-zero data)</td></tr>
<tr><td>1</td><td>ACR (Audio Clock Regeneration)</td></tr>
<tr><td>2</td><td>ASP (Audio Sample Packet)</td></tr>
<tr><td>3</td><td>GCP (General Control Packet)</td></tr>
<tr><td>4</td><td>ACP (Audio Content Protection Packet)</td></tr>
<tr><td>5</td><td>ISRC1 (International Standard Recording Code)</td></tr>
<tr><td>6</td><td>ISRC2 (Internation Standard Recording Code)</td></tr>
<tr><td>7</td><td>OBA (One Bit Audio sample packet)</td></tr>
<tr><td>8</td><td>DTS (DTS Audio packet)</td></tr>
<tr><td>9</td><td>HBR (High Bitrate Audio stream packet)</td></tr>
<tr><td>10</td><td>GMP (Gamut Metadata packet)</td></tr>
<tr><td>11</td><td>3D ASP (3D Audio Sample packet)</td></tr>
<tr><td>12</td><td>3D OBA (3D One Bit Audio sample packet)</td></tr>
<tr><td>13</td><td>AMP (Audio Metadata Packet)</td></tr>
<tr><td>14</td><td>MST_ASP (Multi-stream audio sample packet)</td></tr>
<tr><td>15</td><td>MST_OBA (One Bit Multi-stream audio sample packet)</td></tr>
<tr><td>128:16</td><td>RESERVED</td></tr>
<tr><td>129</td><td>VSI (Vendor Specific Infoframe)</td></tr>
<tr><td>130</td><td>AVI (Auxiliary Video Information)</td></tr>
<tr><td>131</td><td>SPD (Source Product Descriptor)</td></tr>
<tr><td>132</td><td>AIF (Audio Infoframe)</td></tr>
<tr><td>133</td><td>MPEG (MPEG Source infoframe)</td></tr>
<tr><td>134</td><td>RESERVED</td></tr>
<tr><td>135</td><td>DRM (Dynamic Range and Mastering infoframe)</td></tr>
<tr><td>255:136</td><td>RESERVED</td></tr>
</table>

4.4.12.Special purpose configuration items

The configuration items below have one or more special characteristics:

Define	Config ID	Default	Description
TSI_R_GENERIC_STATUS	0x210	N/A	***READ ONLY.*** Report generic device status as status bits.
TSI_R_UNITS_PRESENT	0x211	N/A	***READ ONLY.*** Number of units chained including the master device.
TSI_CURRENT_SINK_EDID	0x1100	N/A	Provides access to A/V input EDID block. ***Important:*** *Not all input types and/or devices allow EDID block block access.*
TSI_VERSION_TEXT	0x80000001	N/A	***READ ONLY.*** Contains text that contains version information of TSI, and of all low-level APIs that are loaded, and of devices that are accessed through each API.
TSI_LOG_FILE	0x80000002	N/A	Char string containing path and file-name of a text file where all TSI status messages are stored. Setting this configuration item will open the target. If the target file already exists, it will be overwritten. To disable status message recording, set this item to empty (i.e. with NULL pointer and zero size). Default: Empty (Log not recorded)
TSI_INVALID_CONFIG_ITEM	0xffffffff	N/A	Invalid configuration ID. Do not use.
TSI_W_ARC_CONTROL	0x1210	0	***WRITE ONLY.*** ARC mode control for devices that support ARC. 0 = Disable ARC. 1 = Enable, Generate audio internally. 2 = Enable, Loop audio back to source.

4.4.13.Configuration items for CRC based video tests.

Define	Config ID	Default	Description
TSI_CRC_TIMEOUT	0x10300	1000	CRC Test max. run time, in milliseconds.
TSI_CRC_FRAMES_TO_TEST	0x10301	20	Number of input frames to be tested. For very long counts, remember to also change the timeout value to allow the frames to be received by TE
TSI_CRC_LIM_FRAME_MISMATCHES	0x10302	0	Number of frames that are allowed to have mismatching CRC.
TSI_CRC_REF_WIDTH	0x10303	1920	Width of the expected video stream coming to the TE, as number of pixels.
TSI_CRC_REF_HEIGHT	0x10304	1080	Height of the expected video stream coming to the TE, as number of pixels.
TSI_CRC_REF_COLORDEPTH	0x10305	24	Number of bits per pixel on the video stream coming to the TE.
TSI_CRC_REFERENCE_CRC_VALUES	0x10306	-	A block of CRC values to be used as references. Each CRC value set is 6 bytes, each with 3 16-bit values, with Red/Cr CRC at lowest address, and Blue/Cb CRC value at highest address.

4.5.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.

4.6.Misc definitions

4.6.1.Option ID codes

Currently no options are implemented. Options are intended to be hardware specific. The intention is that options could be used to modify the behavior of the test equipment.

4.6.2.Data stream type flags

Currently the stream data type flags are defines as follows:

Flag	Description
TSI_FLAG_VIDEO	Video stream is captured from the input.
TSI_FLAG_AUDIO	Audio stream is captured from the input.

The table below defines which devices support which types of streams:

Flag	UCD-1	UCD-2	UCD-323	UFG-6
TSI_FLAG_VIDEO	Supported	Supported	Supported	Supported
TSI_FLAG_AUDIO	Not supported	Not supported	Supported	Supported

5. CONCEPTS

This section explains the concepts used in TSI.

5.1.Parameter ID values

A Parameter ID value defines a specific parameter without defining the size or the type of the parameter. The type and size of content is described in this reference manual for each valid parameter ID value.

5.1.1.Parameter groups

Starting from “client-version 3”, a new type of parameter ID was introduced: parameter groups. Parameter groups are introduced to make GUI development easier, and also to reduce the number of calls needed when enumerating the test parameters. A good example of this are reference frames: A single reference frame requires 8 configuration values to describe a single reference frame.

5.1.2.Developing a dynamic testing GUI

The parameter groups make GUI development easier by enabling creation of GUI component groups that match the parameter groups without needing to filter/search a group of individual parameters that enable using a specific GUI component group. However, this may not be enough: Some parameters are optional for a particular test, while some others are required.