

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **SAI Telemetry and Monitoring** |
| **Authors** | **Broadcom Limited** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **25 January 2017** |
| **SAI-Version** | **1.0** |

Contents

[List of Changes 4](#_Toc473101838)

[1 Overview 6](#_Toc473101839)

[2 Proposal 6](#_Toc473101840)

[2.1 SAI tam Objects 6](#_Toc473101841)

[2.2 Threshold Objects 6](#_Toc473101842)

[2.3 Tracking Mode 6](#_Toc473101843)

[2.4 Snapshot Objects 6](#_Toc473101844)

[2.5 Transporting Snapshots 7](#_Toc473101845)

[3 Specification 7](#_Toc473101846)

[3.1 New file saitam.h 7](#_Toc473101847)

[3.1.1 Data Structures and Enumerations 7](#_Toc473101848)

[3.1.2 API 17](#_Toc473101849)

[3.2 Changes to sai.h 24](#_Toc473101850)

[3.3 Changes to saitypes.h 24](#_Toc473101851)

[4 Examples 24](#_Toc473101852)

[4.1 Setting Up Buffer Tracking 24](#_Toc473101853)

[4.2 Creating and Reading from a Snapshot 25](#_Toc473101854)

[4.3 Creating a Partial Snapshot 26](#_Toc473101855)

[4.4 Creating a Threshold Object 28](#_Toc473101856)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 1 | Proposal for SAI Telemetry and Monitoring | Broadcom Limited | 20 Dec 2016 |
| 2 | Updated to incorporate feedback comments | Broadcom Limited | 25 Jan 2017 |

License

© 2014 Microsoft Corporation, Dell Inc., Facebook, Inc, Broadcom Limited, Intel Corporation, Mellanox Technologies Ltd.

As of September 9, 2014, the following persons or entities have made this Specification available under the Open Web Foundation Final Specification Agreement (OWFa 1.0), which is available at <http://www.openwebfoundation.org/legal/the-owf-1-0-agreements/owfa-1-0>

Microsoft Corporation, Dell Inc., Facebook, Inc, Intel Corporation, Mellanox Technologies Ltd.

You can review the signed copies of the Open Web Foundation Agreement Version 1.0 for this Specification at <http://opencompute.org/licensing/>, which may also include additional parties to those listed above.

Your use of this Specification may be subject to other third party rights. THIS SPECIFICATION IS PROVIDED "AS IS." The contributors expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, noninfringement, fitness for a particular purpose, or title, related to the Specification. The entire risk as to implementing or otherwise using the Specification is assumed by the Specification implementer and user. IN NO EVENT WILL ANY PARTY BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION OR ITS GOVERNING AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE FOLLOWING IS A LIST OF MERELY REFERENCED TECHNOLOGY: Microprocessor technology, semiconductor manufacturing technology, operating system technology (including without limitation networking operating system technology), emulation technology, graphics technology, video technology, integrated circuit packaging technology and the like, compiler technologies, object oriented technology, optical/RF communications technology including chip I/O and driver technology, bus technology, memory chip technology (including, without limitation, NAND memory, NOR memory, resistive RAM (RRAM), seek scan probe (SSP) memory, nonvolatile memory (including without limitation, memory based on chalcogenide materials, phase change memory (PCM), one or more stacked layers of memory cells, embedded PCM memories, non-volatile cache memory, solid state drives, SRAM, embedded DRAM, ferro-electric memory, and polymer memory)) and/or health-related and medical technology. IMPLEMENTATION OF THESE TECHNOLOGIES MAY BE SUBJECT TO THEIR OWN LEGAL TERMS.

# Overview

SAI has provisions for retrieving the values for individual statistics. This document proposes a generic mechanism for snapshots - retrieving statistics data in bulk, as well as introduces a notification mechanism for monitoring statistics using thresholds. The snapshot and notification mechanism introduced in this Specification is generic in nature and can be used by any SAI objects capable of providing statistics and thresholds.

Most Networking Silicon contain a Memory Management Unit which manages packet buffers. The Silicon provides statistics to measure the buffer usage in the MMU. These buffer statistics allow user to get visibility into how the packet buffers are utilized.

# Proposal

## SAI tam Objects

TAM objects provide interface to the buffer trackers on the silicon. They provide API to configure the buffer trackers, setup the tracking modes, creating/deleting buffer statistic snapshots etc. In this spec, these objects are referred to as the **tam** objects.

## Threshold Objects

Networking Silicon may support configuring threshold levels for each of the statistics. Once a threshold is configured for a statistic, the silicon can notify the CPU when the statistic value breaches/exceeds the threshold.

This spec supports threshold configuration via Threshold objects. Threshold objects are created via the **sai\_tam\_threshold\_create\_fn** API. The statistic associated with the threshold object and the associated breach level are passed as attributes during the threshold object creation. A callback function can also be passed as an attribute during the threshold object creation. This callback function will be invoked upon a breach event for the corresponding threshold object.

## Tracking Mode

Networking Silicon may allow tracking either the current, minimum, average or peak values of the statistics. By default, the current values for the statistics are tracked. The required mode can be configured with the **SAI\_TAM\_ATTR\_BUFFER\_TRACKING\_MODE** attribute of the associated **tam** object, using the **sai\_tam\_attribute\_set\_fn** API.

## Snapshot Objects

The Silicon may support taking a current snapshot of all, or selected set, of the statistics, so that they can be used for a comparative analysis. A snapshot may be made by invoking the **sai\_tam\_snapshot\_create\_fn** API. This API returns a snapshot object id, which can be used as an attribute for the **sai\_tam\_counters\_get** API to read all statistics from the snapshot.

Additionally, when a statistic values breaches the configured threshold for that statistic, then the driver automatically takes a snapshot of all the statistics and the snapshot id is provided as part of the event notification function. This behaviour - auto creation of a snapshot when a threshold is breached - can be changed via the attribute **SAI\_TAM\_THRESHOLD\_ATTR\_SNAPSHOT\_ON\_BREACH** of the corresponding threshold object.

Once a snapshot is created, it needs to be explicitly removed. For remote transporters (see below), the snapshots are removed automatically once transport is complete.

## Transporting Snapshots

In general, the snapshot data is made available to the local CPU running the SAI control plane. However, there may be cases where the snapshot data may be required to be sent to a remote host/server. To facilitate such behaviour, the user may specify a transport object (**sai\_tam\_transporter\_t**) to the **tam** object, using the **SAI\_TAM\_ATTR\_TRANSPORTER** attribute.

By default, the **tam** objects copy the snapshots to local CPU. If a transporter is provided, the **tam** object will use the transporter provided, to copy the snapshots. This is applicable for all the snapshots created in the switch, including the ones triggered by a threshold breach.

A user may also specify an alternate transporter for a specific snapshot by using the **SAI\_TAM\_SNAPSHOT\_ATTR\_TRANSPORTER** attribute while creating the snapshot.

It may be noted that the snapshot data may not be available on the local CPU for querying depending on the chosen transporter.

For specifying the remote transporting properties, this spec reuses the SAI *Mirror Session* objects. A Monitor session object is created and is passed as an attribute to the transporter object, during its creation.

# Specification

## New file saitam.h

### Data Structures and Enumerations

/\*\*

\* @brief TAM Statistic ID

\*

\* Identifies a specific counter within the SAI object hierarchy

\* This typedef will be part of saitypes.h and more specifically added to the

\* union sai\_attribute\_value\_t so that this statistic id can be sent as attribute

\* to various API

\*

\*/

typedef struct \_sai\_tam\_statistic\_id\_t {

/\*\*

\* @brief Monitored object id\*/

\*/

sai\_object\_id\_t parent\_id;

/\*\*

\* @brief Monitored object type

\* Ex : SAI\_OBJECT\_TYPE\_BUFFER\_POOL, SAI\_OBJECT\_TYPE\_INGRESS\_PRIORITY\_GROUP

\*/

sai\_object\_type\_t parent\_type;

/\*\*

@brief Counter

\* Ex : SAI\_INGRESS\_PRIORITY\_GROUP\_STAT\_CURR\_OCCUPANCY\_BYTES

\*/

uint64\_t counter\_id;

} sai\_tam\_statistic\_id\_t;

/\*\*

\* @brief TAM Statistic ID List

\*/

typedef struct \_sai\_tam\_statistic\_id\_list\_t {

/\*\*

\* @brief count\*/

\*/

uint32\_t count;

/\*\*

\* @brief list\*/

\*/

sai\_tam\_statistic\_id\_t \*list;

} sai\_tam\_statistic\_id\_list;

/\*\*

\* @brief TAM Statistic

\*

\* Identifies a specific counter within the SAI object hierarchy

\* and provides the current value of the counter

\*/

typedef struct \_sai\_tam\_statistic\_t {

/\*\*

\* @brief Statistic ID\*/

\*/

sai\_tam\_statistic\_id\_t statistic\_id;

/\*\*

\* @brief Value\*/

\*/

uint64\_t value;

} sai\_tam\_statistic\_t;

/\*\*

\* @brief TAM Threshold Breach Event notification

\*/

typedef struct \_sai\_tam\_threshold\_breach\_event\_t {

/\*\*

\* @brief Threshold ID

\*/

sai\_object\_id\_t threshold\_id;

/\*\*

\* @brief Snapshot Valid

\*

\* Indicates whether the snapshot\_id field points to a valid object

\* is\_snapshot\_valid is set to false when the attribute

\* SAI\_TAM\_THRESHOLD\_ATTR\_SNAPSHOT\_ON\_BREACH is either unspecified

\* or set to false.

\*/

bool is\_snapshot\_valid;

/\* @brief Snapshot Id

\* This field is valid only when is\_snapshot\_valid is set to true

\*/

sai\_object\_id\_t snapshot\_id;

/\*\*

\* @brief Threshold / Statistic value for the breach event

\*/

uint64\_t value;

} sai\_tam\_threshold\_breach\_event\_t;

/\*\*

\* TAM Threshold breach notification function

\*/

typedef void(\*sai\_tam\_threshold\_breach\_event\_notification\_fn)(

\_In\_ uint32\_t count,

\_In\_ sai\_tam\_threshold\_breach\_event\_t \*data

);

/\*\*

\* TAM Threshold Attributes.

\*/

typedef enum \_sai\_tam\_threshold\_attr\_t {

/\*\*

\* @brief Start of Attributes

\*/

SAI\_TAM\_THRESHOLD\_ATTR\_START,

/\*\*

\* @brief TAM Object

\*

\* @type sai\_object\_id\_t

\* @flags MANDATORY\_ON\_CREATE

\*/

SAI\_TAM\_THRESHOLD\_ATTR\_TAM\_ID = SAI\_TAM\_THRESHOLD\_ATTR\_START,

/\*\*

\* @brief Statistic for this threshold

\*

\* @type sai\_tam\_statistic\_id\_t

\* @flags MANDATORY\_ON\_CREATE

\*/

SAI\_TAM\_THRESHOLD\_ATTR\_STATISTIC,

/\*\*

\* @brief Level

\*

\* Breach level for this threshold in number of bytes

\*

\* If specified, a threshold breach event will be recorded when the buffer

\* usage goes beyond the level.

\*

\* If not specified, the threshold is created without any level, which is

\* effectively disabling the threshold monitoring for the statistic

\*

\* @type uint64\_t

\* @flags CREATE\_AND\_SET

\* @allownull true

\*/

SAI\_TAM\_THRESHOLD\_ATTR\_LEVEL,

/\*\*

\* @brief Transporter Object

\*

\* Provides the snapshot transporter object for this threshold.

\* When the threshold is breached, this transporter will be used

\* to 'copy' the data to the 'transporter-desired' location.

\*

\* In the absence of a transporter, the tracker's default transporter

\* will be used.

\*

\* It may be noted that, Upon a breach, the 'snapshot' object is

\* automatically created (see below attribute), and it will not have a

\* separate transporter object. Instead this transporter (or the tracker's

\* default transporter) is used.

\*

\* @type sai\_object\_id\_t

\* @allownull true

\*/

SAI\_TAM\_THRESHOLD\_ATTR\_TRANSPORTER,

/\*\*

\* @brief Snapshot on breach

\*

\* Take a snapshot upon a threshold breach.

\* When this attribute is specified and set to true, Snapshots are

\* automatically created upon the threshold breach.

\* Otherwise a snapshot is not created.

\*

\* @type boolean

\* @allownull true

\*/

SAI\_TAM\_THRESHOLD\_ATTR\_SNAPSHOT\_ON\_BREACH,

/\*\*

\* @brief Buffers/Statistics for inclusion in the snapshot

\* Specifies the Statistics/Types for the snapshot.

\* If not specified, all buffers tracked by

\* the associated TAM object are included in the snapshot.

\* When specified, the buffers requested for snapshot must be within the set

\* tracked by the associated TAM object.

\*

\* @type sai\_tam\_statistic\_id\_list

\* @allownull true

\*/

SAI\_TAM\_SNAPSHOT\_ATTR\_STAT\_TYPES,

/\*\*

\* @brief Notification Callback Function

\*

\* Provides the callback function to be invoked upon a thresold breach.

\* In the absence of a callback function, the event will be ignored (DEFAULT)

\*

\* If neither of callback nor transporter is provided, no snapshot is made.

\*

\* If callback is required but SAI\_TAM\_THRESHOLD\_ATTR\_SNAPSHOT\_ON\_BREACH is

\* either unspecified, or set to false, then the event data passed to the

\* callback function will have the field is\_snapshot\_valid set to false.

\*

\* @type sai\_tam\_threshold\_breach\_event\_notification\_fn

\* @allownull true

\*/

SAI\_TAM\_THRESHOLD\_ATTR\_BREACH\_CALLBACK,

/\*\*

\* @brief End of Attributes

\*/

SAI\_TAM\_TAMTHRESHOLD\_\_ATTR\_END

} sai\_tam\_threshold\_attr\_t;

/\*\*

\* TAM Snapshot Attributes.

\*/

typedef enum \_sai\_tam\_snapshot\_attr\_t {

/\*\*

\* @brief Start of Attributes

\*/

SAI\_TAM\_SNAPSHOT\_ATTR\_START,

/\*\*

\* @brief TAM Object for this snapshot

\*

\* Specifies the TAM object for this snapshot.

\*

\* @type sai\_object\_id\_t

\* @flags MANDATORY\_ON\_CREATE

\*/

SAI\_TAM\_SNAPSHOT\_ATTR\_TAM\_ID = SAI\_TAM\_SNAPSHOT\_ATTR\_START,

/\*\*

\* @brief Buffers/Statistics for inclusion in snapshot

\*

\* Specifies the Statistics/Types for a snapshot.

\* If not specified, all buffers tracked by

\* the associated TAM object are included in the snapshot. (DEFAULT)

\* When specified, the buffers requested for snapshot must be within the set

\* tracked by the associated TAM object.

\*

\* @type sai\_tam\_statistic\_id\_list\_t

\* @flags CREATE\_AND\_SET

\* @allownull true

\*/

SAI\_TAM\_SNAPSHOT\_ATTR\_STAT\_TYPE,

/\*\*

\* @brief Transporter Object

\*

\* Provides the snapshot transporter object for this snapshot.

\* When the snapshot is made, this transporter will be used

\* to 'copy' the data to the 'transporter-desired' location.

\* In the absence of a transporter, the tracker's default transporter

\* will be used (DEFAULT)

\*

\* @type sai\_object\_id\_t

\* @flags CREATE\_AND\_SET

\* @allownull true

\*/

SAI\_TAM\_SNAPSHOT\_ATTR\_TRANSPORTER,

/\*\*

\* @brief End of Attributes

\*/

SAI\_TAM\_SNAPSHOT\_ATTR\_END

} sai\_tam\_snapshot\_attr\_t;

/\*\*

\* @brief TAM Tracking Options

\*/

typedef enum \_sai\_tam\_tracking\_options\_t {

SAI\_TAM\_TRACKING\_MODE\_PEAK,

SAI\_TAM\_TRACKING\_MODE\_CURRENT,

SAI\_TAM\_TRACKING\_MODE\_AVERAGE,

SAI\_TAM\_TRACKING\_MODE\_MINIMUM

} sai\_tam\_tracking\_options\_t;

/\*\*

\* @brief TAM Reporting Options

\*/

typedef enum \_sai\_tam\_reporting\_options\_t {

SAI\_TAM\_REPORTING\_MODE\_BYTES,

SAI\_TAM\_REPORTING\_MODE\_PERCENTAGE,

} sai\_tam\_reporting\_options\_t;

/\*\*

\* @ TAM Attributes.

\*/

typedef enum \_sai\_tam\_attr\_t {

/\*\*

\* @brief Start of Attributes

\*/

SAI\_TAM\_ATTR\_START,

/\*

\* @brief Switch Object for this TAM

\*

\* @type sai\_object\_id\_t

\* @flags MANDATORY\_ON\_CREATE

\*/

SAI\_TAM\_ATTR\_SWITCH\_ID= SAI\_TAM\_ATTR\_START,

/\*

\* @brief Total Number of counters supported

\*

\* @type uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_TAM\_ATTR\_TOTAL\_NUM\_STATISTICS,

/\*

\* @brief Latest Snapshot ID

\*

\* @type sai\_object\_id\_t

\* @flags READ\_ONLY

\*/

SAI\_TAM\_ATTR\_LATEST\_SNAPSHOT\_ID,

/\*

\* @brief Maximum Number of snapshots that can be created.

\* If the number of currently created snapshots already reach this limit, any

\* attempt to create more snapshots return error.

\*

\* @type uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_TAM\_ATTR\_MAX\_NUM\_SNAPSHOTS,

/\*

\* @brief Clear all Thresholds

\* If this attribute is specified and set to true, then

\* the following actions take place for each of the created threshold objects

\* 1. The values specified via the SAI\_TAM\_THRESHOLD\_ATTR\_LEVEL attribute are

\* removed

\* 2. Threshold monitoring is disabled

\*

\* @type Boolean

\* @flags WRITE\_ONLY | CREATE\_AND\_SET

\*/

SAI\_TAM\_ATTR\_CLEAR\_ALL\_THRESHOLDS,

/\*

\* @brief Operational State for the Buffer Tracking

\*

\* @type boolean

\* @default true

\* @allownull true

\*/

SAI\_TAM\_ATTR\_BUFFER\_TRACKING\_ADMIN\_STATE,

/\*

\* @brief Statistics reporting mode.

\* When not specified, reports in number of bytes (DEFAULT)

\*

\* @type sai\_tam\_reporting\_options\_t

\* @default SAI\_TAM\_REPORTING\_MODE\_BYTES

\* @allownull true

\*/

SAI\_TAM\_ATTR\_BUFFER\_REPORTING\_MODE,

/\*\*

\* @brief Buffer Tracker Mode

\*

\* Specifies whether the Chip should track the peak values of the

\* buffers or current usage values (DEFAULT)

\*

\* @type sai\_tam\_tracking\_options\_t

\* @default SAI\_TAM\_TRACKING\_MODE\_CURRENT

\* @allownull true

\*/

SAI\_TAM\_ATTR\_BUFFER\_TRACKING\_MODE,

/\*\*

\* @brief Buffers/Statistics for tracking using this object

\*

\* Specifies the Statistics/Types for tracking. If not specified, all

\* supported buffers are included for tracking. (DEFAULT)

\*

\* A statistic can't be tracked by more then one TAM Objects

\*

\* @type sai\_tam\_statistic\_id\_list\_t

\* @allownull true

\*/

SAI\_TAM\_ATTR\_TRACKING\_OPTIONS,

/\*\*

\* @brief Default Transporter

\*

\* Provides a default snapshot transporter object for the Tracker.

\* When a snapshot is made, this transporter will be used

\* to 'copy' the data to the 'transporter-desired' location.

\*

\* In the absence of a transporter, the tracker will copy the

\* data to the local CPU transporter.

\*

\* @type sai\_object\_id\_t

\* @allownull true

\*/

SAI\_TAM\_ATTR\_TRANSPORTER,

/\*\*

\* @brief End of Attributes

\*/

SAI\_TAM\_ATTR\_END

} sai\_tam\_attr\_t;

/\*\*

\* @brief TAM type of snapshot transport

\*/

typedef enum \_sai\_tam\_transporter\_type\_t {

/\*\* Local, to the CPU \*/

SAI\_TAM\_TRANSPORTER\_TYPE\_LOCAL,

/\*\* Remote, to a remote monitoring client \*/

SAI\_TAM\_TRANSPORTER\_TYPE\_REMOTE,

} sai\_tam\_transporter\_type\_t;

/\*\*

\* @brief TAM Snapshot Transporter Attributes

\*/

typedef enum \_sai\_tam\_transporter\_attr\_t {

/\*\*

\* @brief Start of Attributes

\*/

SAI\_TAM\_TRANSPORTER\_ATTR\_START,

/\*\*

\* @brief Transporter Type

\*

\* If this attribute value is unspecified the local transporter is used.

\*

\* @type sai\_tam\_transporter\_type\_t

\* @allownull true

\*/

SAI\_TAM\_TRANSPORTER\_ATTR\_TYPE = SAI\_TAM\_TRANSPORTER\_ATTR\_START,

/\*\*

\* @brief Maximum size beyond which it will be truncated.

\*

\* If this attribute value is zero or unspecified, snapshots are not

\* truncated while transporting. (DEFAULT)

\*

\* @type uint32\_t

\* @allownull true

\*

\*/

SAI\_TAM\_TRANSPORTER\_ATTR\_MAX\_SNAPSHOT\_SIZE,

/\*\*

\* @type Mirroring session object defining the remote transport capabilities.

\*

\* If this attribute is unspecified, Local CPU Transport is used (DEFAULT)

\*

\* @type sai\_object\_id\_t

\* @allownull true

\*

\*/

SAI\_TAM\_TRANSPORTER\_ATTR\_MONITOR\_ID,

/\*\*

\* @brief End of Attributes

\*/

SAI\_TAM\_TRANSPORTER\_ATTR\_END

} sai\_tam\_transporter\_attr\_t;

### API

/\*\*

\* Routine Description:

\* @brief Create and return a TAM object

\* This creates a TAM object in the driver for tracking the buffer usage.

\* Via the attributes, caller may indicate a preference for tracking of a

\* specific set of statistics/groups.

\* Arguments:

\* @param[out] tam\_id - TAM object

\* @param[in] attr\_list – tam attributes

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_create\_fn) (

\_Out\_ sai\_object\_id\_t \* tam\_id,

\_In\_ uint32\_t attr\_count,

\_In\_ sai\_attribute\_t \* attr\_list /\* sai\_tam\_attr\_t \*/

);

/\*\*

\* Routine Description:

\*

\* @brief Deletes a specified tam object.

\*

\* Deleting a tam object also deletes all associated snapshot and threshold objects.

\*

\* Arguments:

\* @param[in] tam\_id - tam object to be removed.

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_remove\_fn) (

\_In\_ sai\_object\_id\_t \*tam\_id

);

/\*\*

\* Routine Description:

\* @brief Get values for specified TAM attributes.

\*

\* Arguments:

\* @param[in] tam\_id – tam object id

\* @param[in] attr\_count - number of attributes

\* @param[inout] attr\_list - array of attributes

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_attribute\_get\_fn) (

\_In\_ sai\_object\_id\_t tam\_id,

\_In\_ uint32\_t attr\_count,

\_Inout\_ sai\_attribute\_t \*attr\_list

);

/\*\*

\* Routine Description:

\* @brief Set TAM attribute value(s).

\*

\* Arguments:

\* @param[in] tam\_id - tam id

\* @param[in] attr - attribute

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_attribute\_set\_fn) (

\_In\_ sai\_object\_id\_t tam\_id,

\_In\_ uint32\_t attr\_count,

\_In\_ sai\_attribute\_t \*attr\_list

);

/\*\*

\* Routine Description:

\* @brief Create and return a snapshot object

\*

\* This creates a snapshot in the hardware and copies the snapshot data

\* into the driver. Via the attributes, caller may indicate a preference

\* for snapshot of a specific set of statistics/groups.

\*

\* Arguments:

\* @param[out] snapshot\_id - snapshot object

\* @param[in] attr\_list – [sai\_tam\_attribute\_t]

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_snapshot\_create\_fn) (

\_Out\_ sai\_object\_id\_t \* snapshot\_id,

\_In\_ uint32\_t attr\_count,

\_In\_ sai\_attribute\_t \* attr\_list /\* sai\_tam\_snapshot\_attr\_t \*/

);

/\*\*

\* Routine Description:

\* @brief Deletes a specified snapshot object and free driver memory.

\*

\* Arguments:

\* @param[in] snapshot\_id - snapshot object to be removed.

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_snapshot\_remove\_fn) (

\_In\_ sai\_object\_id\_t \* snapshot\_id

);

/\*\*

\* Routine Description:

\* @brief Get values for specified Snapshot attributes.

\*

\* Arguments:

\* @param[in] snapshot\_id – snapshot object id

\* @param[in] attr\_count - number of attributes

\* @param[inout] attr\_list - array of attributes

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_snapshot\_attribute\_get\_fn) (

\_In\_ sai\_object\_id\_t snapshot\_id,

\_In\_ uint32\_t attr\_count,

\_Inout\_ sai\_attribute\_t \* attr\_list

);

/\*\*

\* Routine Description:

\* @brief Set Snapshot attribute value(s).

\*

\* Arguments:

\* @param[in] snapshot\_id - snapshot object id

\* @param[in] attr\_list - attribute

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_snapshot\_attribute\_set\_fn) (

\_In\_ sai\_object\_id\_t snapshot\_id,

\_In\_ uint32\_t attr\_count,

\_In\_ sai\_attribute\_t \* attr\_list

);

/\*\*

\* Routine Description :

\* @brief Obtain the values for all statistics from a snapshot.

\*

\* Attribute list must supply sufficient memory for statistics

\* as specified for the snapshot object, which may be all statistics

\* supported by the associated tam object.

\*

\* Arguments:

\* @param[in] snapshot\_id – snapshot object id

\* @param[inout] stat\_count - number of statistics (required/provided)

\* @param[inout] statistics - statistics (allocated/provided)

\*

\* Return Values :

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_snapshot\_statistics\_get\_fn) (

\_In\_ sai\_object\_id\_t snapshot\_id,

\_Inout\_ uint32\_t \* stat\_count,

\_Inout\_ sai\_tam\_statistic\_t \* statistics

);

/\*\*

\* Routine Description:

\* @brief Create and return a threshold object

\*

\* This creates a threshold in the hardware with the associated statistic

\* passed via the attributes.

\*

\* Arguments:

\* @param[out] threshold\_id - threshold object

\* @param[in] attr\_list – preferences for creating a threshold

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_threshold\_create\_fn) (

\_Out\_ sai\_object\_id\_t \* threshold\_id,

\_In\_ uint32\_t attr\_count,

\_In\_ sai\_attribute\_t \* attr\_list /\* sai\_tam\_threshold\_attr\_t \*/

);

/\*\*

\* Routine Description:

\* @brief Deletes a specified threshold object.

\*

\* Arguments:

\* @param[in] threshold\_id - threshold object to be removed.

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_threshold\_remove\_fn) (

\_In\_ sai\_object\_id\_t \* threshold\_id

);

/\*\*

\* Routine Description:

\* @brief Get values for specified threshold attributes.

\*

\* Arguments:

\* @param[in] threshold\_id – threshold object id

\* @param[in] attr\_count - number of attributes

\* @param[inout] attr\_list - array of attributes

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_threshold\_attribute\_get\_fn) (

\_In\_ sai\_object\_id\_t threshold\_id,

\_In\_ uint32\_t attr\_count,

\_Inout\_ sai\_attribute\_t \* attr\_list

);

/\*\*

\* Routine Description:

\* @brief Set threshold attribute value(s).

\*

\* Arguments:

\* @param[in] threshold\_id - threshold object id

\* @param[in] attr\_list - attribute

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_threshold\_attribute\_set\_fn) (

\_In\_ sai\_object\_id\_t threshold\_id,

\_In\_ uint32\_t attr\_count,

\_In\_ sai\_attribute\_t \* attr\_list

);

/\*\*

\* Routine Description:

\* @brief Create and return a Transporter object

\*

\* This creates a transport object for copying the snapshot data

\* to the desired location

\*

\* Arguments:

\* @param[out] transporter\_id - Transporter object

\* @param[in] attr\_list – tam trackers for creating a transporter

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_transporter\_create\_fn) (

\_Out\_ sai\_object\_id\_t \* transporter\_id,

\_In\_ uint32\_t attr\_count,

\_In\_ sai\_attribute\_t \* attr\_list /\* sai\_tam\_transporter\_attr\_t \*/

);

/\*\*

\* Routine Description:

\* @brief Deletes a specified Transporter object.

\*

\* Arguments:

\* @param[in] transporter\_id - Transporter object to be removed.

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_transporter\_remove\_fn) (

\_In\_ sai\_object\_id\_t \* transporter\_id

);

/\*\*

\* Routine Description:

\* @brief Get values for specified Transporter attributes.

\*

\* Arguments:

\* @param[in] transporter\_id – Transporter object id

\* @param[in] attr\_count - number of attributes

\* @param[inout] attr\_list - array of attributes

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_transporter\_attribute\_get\_fn) (

\_In\_ sai\_object\_id\_t transporter\_id,

\_In\_ uint32\_t attr\_count,

\_Inout\_ sai\_attribute\_t \*attr\_list

);

/\*\*

\* Routine Description:

\* @brief Set TAM Transporter attribute value(s).

\*

\* Arguments:

\* @param[in] transporter\_id - transporter object id

\* @param[in] attr - attribute

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_tam\_transporter\_attribute\_set\_fn) (

\_In\_ sai\_object\_id\_t transporter\_id,

\_In\_ uint32\_t attr\_count,

\_In\_ sai\_attribute\_t \* attr\_list

);

typedef struct \_sai\_tam\_api\_t {

/\* TAM Object API \*/

sai\_tam\_create\_fn tam\_create;

sai\_tam\_remove\_fn tam\_remove;

sai\_tam\_attribute\_set\_fn tam\_attribute\_set;

sai\_tam\_attribute\_get\_fn tam\_attribute\_get;

/\* Snapshot object API \*/

sai\_tam\_snapshot\_create\_fn tam\_snapshot\_create;

sai\_tam\_snapshot\_remove\_fn tam\_snapshot\_remove;

sai\_tam\_snapshot\_attribute\_set\_fn tam\_snapshot\_attribute\_set;

sai\_tam\_snapshot\_attribute\_get\_fn tam\_snapshot\_attribute\_get;

sai\_tam\_snapshot\_statistics\_get\_fn tam\_snapshot\_statistics\_get;

/\* Threshold object API \*/

sai\_tam\_threshold\_create\_fn tam\_threshold\_create;

sai\_tam\_threshold\_remove\_fn tam\_threshold\_remove;

sai\_tam\_threshold\_attribute\_set\_fn tam\_threshold\_attribute\_set;

sai\_tam\_threshold\_attribute\_get\_fn tam\_threshold\_attribute\_get;

/\* Transporter Object API \*/

sai\_tam\_transporter\_create\_fn tam\_transporter\_create;

sai\_tam\_transporter\_remove\_fn tam\_transporter\_remove;

sai\_tam\_transporter\_attribute\_set\_fn tam\_transporter\_attribute\_set;

sai\_tam\_transporter\_attribute\_get\_fn tam\_transporter\_attribute\_get;

} sai\_tam\_api\_t;

## Changes to sai.h

typedef enum \_sai\_api\_t {

…

SAI\_API\_TAM = 0x200, /\* sai\_tam\_api\_t \*/

}

## Changes to saitypes.h

The following are the changes to the saitypes.h file.

* The union **sai\_object\_type\_t** is enhanced to include the following new object types:
  + **SAI\_OBJECT\_TYPE\_TAM**
  + **SAI\_OBJECT\_TYPE\_TAM\_SNAPSHOT**,
  + **SAI\_OBJECT\_TYPE\_TAM\_TRANSPORTER,**
  + **SAI\_OBJECT\_TYPE\_TAM\_THRESHOLD**
* The union **sai\_attribute\_value\_t** is enhanced to include the following new object types:
  + **sai\_tam\_statistic\_id\_t (field name statistic\_id)**
  + **sai\_tam\_statistic\_id\_list (field name statistic\_id\_list)**

# Examples

## Setting Up Buffer Tracking

sai\_object\_id\_t tam\_id;

sai\_object\_id\_t switch\_id;

sai\_attribute\_t tam\_attrs[5];

sai\_attribute\_t transporter\_attrs[1];

sai\_object\_t transporter\_id;

/\* ElseWhere -- Get the switch Object into switch\_id \*/

/\* Create the TAM object for all statistics \*/

/\* Create the Local Transport \*/

transporter\_attrs[0].id = (sai\_attr\_id\_t)SAI\_TAM\_TRANSPORTER\_ATTR\_TYPE;

transporter\_attrs[0].s32 = SAI\_TAM\_TRANSPORTER\_TYPE\_LOCAL;

tam\_transporter\_create(&transporter\_id, 1, &transporter\_attrs[0]);

/\* Enable TAM Tracking \*/

tam\_attrs[0].id = (sai\_attr\_id\_t)SAI\_TAM\_ATTR\_BUFFER\_TRACKING\_ADMIN\_STATE;

tam\_attrs[0].value.booldata = 1;

/\* let the reporting be in percentages \*/

tam\_attrs[1].id = (sai\_attr\_id\_t)SAI\_TAM\_ATTR\_BUFFER\_REPORTING\_MODE;

tam\_attrs[1].value.s32 = SAI\_TAM\_REPORTING\_MODE\_PERCENTAGE;

/\* we want peak values of buffer usage rather than current usage \*/

tam\_attrs[2].id = (sai\_attr\_id\_t)SAI\_TAM\_ATTR\_BUFFER\_TRACKING\_MODE;

tam\_attrs[2].value.s32 = SAI\_TAM\_TRACKING\_MODE\_PEAK;

/\* Switch \*/

tam\_attrs[3].id = (sai\_attr\_id\_t)SAI\_TAM\_ATTR\_SWITCH\_ID;

tam\_attrs[3].value.oid = switch\_id;

/\* Transport \*/

tam\_attrs[4].id = (sai\_attr\_id\_t)SAI\_TAM\_ATTR\_TRANSPORTER;

tam\_attrs[4].value.oid = transporter\_id;

/\* create the TAM Object \*/

sai\_tam\_create(&tam\_id, 5, &tam\_attrs[0]);

## Creating and Reading from a Snapshot

int num\_counters = 0;

sai\_tam\_statistic\_t \*stat\_list;

sai\_attribute\_t attrs[3];

sai\_object\_id\_t snapshot\_id;

sai\_object\_id\_t tam\_id;

/\* Obtain TAM object \*/

...

/\* find out the number of available statistics \*/

attrs[0].id = (sai\_attr\_id\_t) SAI\_TAM\_ATTR\_NUM\_STATISTICS;

tam\_attribute\_get(tam\_id, 1, &attrs[0]);

num\_counters = attrs[0].value.s32;

/\* allocate host memory for 'num\_counters' of 'sai\_tam\_statistic\_t' counters in 'stat\_list’ \*/

...

/\* create a complete snapshot \*/

attrs[0].id = (sai\_attr\_id\_t)SAI\_TAM\_SNAPSHOT\_ATTR\_TAM\_ID;

attrs[0].value.oid = tam\_id;

sai\_tam\_snapshot\_create(&snapshot\_id, 1, &attrs[0]);

/\* read from snapshot \*/

sai\_tam\_snapshot\_statistics\_get(snapshot\_id, &num\_counters , stat\_list );

/\* delete snapshot \*/

sai\_tam\_snapshot\_remove(snapshot\_id);

## Creating a Partial Snapshot

/\* create a partial snapshot\*/

sai\_attribute\_t tam\_attrs[5];

sai\_object\_id\_t snapshot\_id;

sai\_object\_id\_t tam\_id;

sai\_object\_id\_t ippg\_port3\_pg1\_id;

sai\_object\_id\_t ippg\_port3\_pg2\_id;

sai\_object\_id\_t ippg\_port3\_pg3\_id;

sai\_object\_id\_t ippg\_port3\_pg4\_id;

/\* Memory for reading a snapshot \*/

sai\_tam\_statistic\_t \*stat\_list;

int num\_counters = 0;

/\* Statistic Id list , for creating a partial snapshot \*/

sai\_tam\_statistic\_id\_list stat\_id\_list;

sai\_tam\_statistic\_id\_t stat\_id\_array[4];

/\* Obtain TAM object \*/

...

tam\_attrs[0].id = (sai\_attr\_id\_t)SAI\_TAM\_SNAPSHOT\_ATTR\_TAM\_ID;

tam\_attrs[0].value.oid = tam\_id;

/\*

Interested in Ingress-Port-Priority-Group Statistics for port 3 and all priority-groups for the current-occupancy counter.

Obtain priority group objects ids from priority group list attribute on port 3.

Caller/Application is assumed to have created the relevant priority-group objects earlier and the associated object ids are used here (in this case, ippg\_port3\_pg1\_id etc).

\*/

...

stat\_id\_array [0].parent\_id = ippg\_port3\_pg1\_id;

stat\_id\_array [0].parent\_type = SAI\_OBJECT\_TYPE\_INGRESS\_PRIORITY\_GROUP;

stat\_id\_array [0].counter\_id = SAI\_INGRESS\_PRIORITY\_GROUP\_STAT\_CURR\_OCCUPANCY\_BYTES;

stat\_id\_array [1].parent\_id = ippg\_port3\_pg2\_id;

stat\_id\_array [1].parent\_type = SAI\_OBJECT\_TYPE\_INGRESS\_PRIORITY\_GROUP;

stat\_id\_array [1].counter\_id = SAI\_INGRESS\_PRIORITY\_GROUP\_STAT\_CURR\_OCCUPANCY\_BYTES;

stat\_id\_array [2].parent\_id = ippg\_port3\_pg3\_id;

stat\_id\_array [2].parent\_type = SAI\_OBJECT\_TYPE\_INGRESS\_PRIORITY\_GROUP;

stat\_id\_array [2].counter\_id = SAI\_INGRESS\_PRIORITY\_GROUP\_STAT\_CURR\_OCCUPANCY\_BYTES;

stat\_id\_array [3].parent\_id = ippg\_port3\_pg4\_id;

stat\_id\_array [3].parent\_type = SAI\_OBJECT\_TYPE\_INGRESS\_PRIORITY\_GROUP;

stat\_id\_array [3].counter\_id = SAI\_INGRESS\_PRIORITY\_GROUP\_STAT\_CURR\_OCCUPANCY\_BYTES;

/\* create the required list of statistic ids \*/

stat\_id\_list.count = 4;

stat\_id\_list.list = &stat\_id\_array[0];

tam\_attrs[1].id = (sai\_attr\_id\_t) SAI\_TAM\_SNAPSHOT\_ATTR\_STAT\_TYPE;

tam\_attrs[1].value. statistic\_id\_list = stat\_id\_list;

/\* create the required partial snapshot \*/

sai\_tam\_snapshot\_create(&snapshot\_id, tam\_id, 2, &tam\_attrs[0]);

/\* find out the number of available statistics \*/

tam\_attrs[0].id = (sai\_attr\_id\_t) SAI\_TAM\_ATTR\_NUM\_STATISTICS;

tam\_attribute\_get(tam\_id, 1, &tam\_attrs[0]);

num\_counters = attrs[0].value.s32;

/\* allocate host memory for 'num\_counters' of 'sai\_tam\_statistic\_t' counters in 'stat\_list’ \*/

/\* read from snapshot \*/

sai\_tam\_snapshot\_statistics\_get(snapshot\_id, &num\_counters , stat\_list );

/\* delete snapshot \*/

sai\_tam\_snapshot\_remove(snapshot\_id);

## Creating a Threshold Object

sai\_attribute\_t attrs[5];

sai\_object\_id\_t threshold\_id;

sai\_object\_id\_t tam\_id;

sai\_tam\_statistic\_id\_t stat;

sai\_object\_id\_t ippg\_port3\_pg1\_id;

sai\_tam\_threshold\_breach\_event\_notification\_fn mycallback;

/\* The following callback defined elsewhere \*/

void my\_tam\_threshold\_breach\_event\_notification\_cb(

\_In\_ uint32\_t count,

\_In\_ sai\_tam\_threshold\_breach\_event\_t \*data

);

mycallback = (sai\_tam\_threshold\_breach\_event\_notification\_fn) my\_tam\_threshold\_breach\_event\_notification\_cb;

/\* Obtain TAM object id. This object has a default transporter defined. \*/

...

/\*

Interested in Ingress-Port-Priority-Group Statistics for port 3. Set threshold for desired priority-group value of the current-occupancy counter.

\*/

stat.parent\_id = ippg\_port3\_pg1\_id;

stat.parent\_type = SAI\_OBJECT\_TYPE\_INGRESS\_PRIORITY\_GROUP;

stat.counter\_id = SAI\_INGRESS\_PRIORITY\_GROUP\_STAT\_CURR\_OCCUPANCY\_BYTES;

/\* Create a threshold object that sends a snapshot to the default transporter.\*/

attrs[0].id = (sai\_attr\_id\_t)SAI\_TAM\_THRESHOLD\_ATTR\_TAM\_ID;

attrs[0].value.oid = tam\_id;

attrs[1].id = (sai\_attr\_id\_t)SAI\_TAM\_THRESHOLD\_ATTR\_STATISTIC;

attrs[1].value.statistic\_id = stat;

attrs[2].id = (sai\_attr\_id\_t)SAI\_TAM\_THRESHOLD\_ATTR\_LEVEL;

attrs[2].value.uint64\_t = 50000;

attrs[3].id = (sai\_attr\_id\_t)SAI\_TAM\_THRESHOLD\_SNAPSHOT\_ON\_BREACH;

attrs[3].value.bool = true;

attrs[4].id = (sai\_attr\_id\_t) SAI\_TAM\_THRESHOLD\_ATTR\_BREACH\_CALLBACK;

attrs[4].value.ptr = (sai\_pointer\_t)mycallback;

sai\_tam\_threshold\_create(&threshold\_id, 5, &attrs[0]);