Configuration & Status API Programming Guide

EDM04-08

Protection Against Harmful Interference

When present on equipment this manual pertains to, the statement "This device complies with part 15 of the FCC rules" specifies the equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the Federal Communications Commission [FCC] Rules.

These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment.

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications.

Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at their own expense.

Extra Components and Materials

The product that this manual pertains to may include extra components and materials that are not essential to its basic operation, but are necessary to ensure compliance to the product standards required by the United States Federal Communications Commission, and the European EMC Directive. Modification or removal of these components and/or materials, is liable to cause non compliance to these standards, and in doing so invalidate the user's right to operate this equipment in a Class A industrial environment.

Disclaimer

Website

http://www.endace.com

Copyright 2008 Endace Technology Ltd. All rights reserved.

Contents

Introduction 1

Overview 1
Version 1
Supported Cards 1
Purpose 1
Thread Safety 2
Support 2

Chapter 2: API Overview 3

Overview 3
Components 3
Card Configuration 3
Example Program 5
Using the API 6

Chapter 3: Components and Attributes 7

Overview 7
Your Card's Components and Attributes 7
Component Definitions 8
Components per card 10
kComponentCardInfo 12
kComponentConnectionSetup 12
kComponentCounter 12
kComponentDemapper (channelized) 13
KComponentDemapper (concatenated) 13
kComponentDrop 13
kComponentDSM 14
kComponentDUCK 14
kComponentE1T1 15
kComponentErfMux 15
kComponentFramer 15
kComponentGeneral 15
kComponentGpp 16
kComponentHardwareMonitor 16
kComponentInterface 16
kComponentIPF 17
kComponentLEDController 17
kComponentMapper (3.7T) 17
kComponentMapper (channelized) 18
kComponentMapper (concatenated) 18
kComponentMiniMacStatistics 19
kComponentMux 20
kComponentOptics 20
kComponentPbm 21
kComponentPCS 21
kComponentPhy 22
kComponentPort 23
kComponentSC256 30
kComponentSonetPP 30
kComponentSonic 31
kComponentSteering 32
kComponentStream 32
kComponentStreamFeatures 33
kComponentTerf 33

kComponentXGMII 33
Chapter 4: DAG 3.7T Card 35
Structures 35
Notes on the Connection Components 37
Layout: 37
Mode Table 38
Enumerations 39
Chapter 5: DAG7.1S Card 40
Structures 40
Chapter 6: Card Configuration Functions 41
Overview 41
Functions 41
Chapter 7: Component Functions 47
Overview 47
Functions 47
Chapter 8: Attribute Accessor Functions 51
Overview 51
Functions 51
Chapter 9: Modifier Functions 53
Overview 53
Functions 53
Chapter 10: Firmware Functions 55
Overview 55
Functions 55
Chapter 11: Data Structures and Constants 57
Overview 57
Types 57
Version History 69

Introduction

Overview

The Endace range of DAG cards provides the means to transfer data at the full speed of a network into the memory of a host PC, with zero packet loss in even the worst-case conditions.

The present range of Endace Accelerated™ card products provide the benefits of Endace technology across the broad array of contemporary network standards, ranging from legacy copper T1/E1 through to modern high-speed optical OC192/STM-64 and 10G Ethernet.

The Endace Configuration and Status Application Programming Interface (API) enables developers to configure the varied range of components and associated attributes of a DAG card. Previously this was only possible using the individual DAG command line utilities.

It allows allow third-party developers to perform the following tasks from within their own application software:

- Resetting a DAG card.
- Loading firmware images onto a DAG card.
- Setting and retrieving the hardware configuration.
- Retrieving status and statistics information.

Version

The information in this document is correct for DAG software version 3.2.1. Please see the release notes for your software version for a list of supported DAG cards.

Supported Cards

The Configuration & Status API supports the following Endace Accelerated™ DAG cards:

- DAG 3.7D
- DAG 3.7GE
- DAG 3.7T
- DAG 3.8S
- DAG 4.3S
- DAG 4.3GE
- DAG 4.5G2/G4
- DAG 5.0SG2/SG2A
- DAG 5.2X
- DAG 5.2SXA
- DAG 5.4 / 5.4A
- DAG 8.1SX
- DAG 8.1X
- DAG 8.2X
- DAG 8.4I

Please refer to the relevant DAG Card User Guide available from the Endace Technical Support website at http://www.endace.com/support for detailed information on the features and functionality of each DAG card.

Purpose

The purpose of this Programming Guide is to:

- Provide you with general information about the Configuration & Status API
- Describe the components and attributes associated with each of the DAG cards supported by the API
- Define function definitions
- Describe data structures and constants

Thread Safety

Please note that the routines described in this Programming Guide are not thread safe or re-entrant. If you are using multiple threads Endace strongly recommends that you use wrapper functions to serialize access to the Endace supplied routines.

Support

Endace Website

In the event that you experience problems with any Endace supplied hardware, or software, it is recommended that you visit the Endace website at http://www.endace.com. This website includes a *Support* page which offers a range of online assistance options including a Public Knowledge Base. It also allows you to submit a problem report online via the *Online Case Submission* link.

If you have a support contract with Endace you can login using your support username and password which provides access to the secure area of the website. This contains the latest versions of software, device drivers, firmware, user manuals, and release notes.

For more information about the Endace Support Package, or how to obtain (or change) your secure support website login details, please contact sales@endace.com

If you are unable to resolve a problem using the information on the website, you can email Endace Technical Support at support@endace.com for further assistance.

Reporting Problems

When reporting a problem please supply as much information as possible. The more information you supply the quicker Endace Technical Support will be able to effectively respond to you. Although the exact information available may be limited by the type of problem you are experiencing, you should try to supply the following:

- Contents of any scripts in use.
- DAG card model and serial number.
- DAG software version in use as returned by `rpm -q dag-base`
- System log messages generated when DAG device driver is loaded. These can be collected from command `dmesg`, or from log file `/var/log/syslog`.
- Output of `daginf`.
- Firmware versions from `dagrom -x`
- Card configuration as reported by: `dagconfig`
- Network link statistics reported by: `dagconfig -ei`
- Network link configuration from the router where available.
- Complete output of session where error occurred including any error messages from DAG tools. The `typescript` Unix utility may be useful for this.
- A small section of captured packet trace illustrating the problem.

Chapter 2: API Overview

Overview

The Endace Configuration and Status Application Programming Interface (API) enables developers to configure the components and associated attributes of an Endace Data Acquisition and Generation (DAG) card.

It allows third-party developers to perform the following tasks from within their own application software:

- Resetting a DAG card.
- Loading firmware images onto a DAG card.
- Setting and retrieving the hardware configuration.
- Retrieving status and statistics information.

Components

The main processing unit of each DAG card resides inside a *Field Programmable Gate Arrays* (FPGA) chip. This system consists of several *modules* that process the input data in parallel. Each of these modules can have several *configurations* (not to confuse with the SRAM configuration of the FPGA). In order to bring a module into a required configuration you need to write appropriate control data into various registers inside the FPGA. The Configuration and Status API provides a high-level method of accessing these registers and thereby allows you to control the behavior of the DAG card within your C or C++ programs.

The model of a DAG card implied by the API is a hierarchical tree of *components* where these components correspond to the modules inside the FPGA (such as packet processor, PCI burst manager etc). The various other chips on the board such as physical interfaces and hardware monitors are also controlled through the FPGA and as such have corresponding components. The top component in the tree is called the *root* component which contains a reference to the attached DAG card. Each subcomponent of the root has a set of attributes associated with it which defines the configuration of the module at any point in time. Changing the value of the component attributes directly changes the behavior of the corresponding modules. This document describes all components and attributes on a per card basis.

Note: Not all components and attributes are common to all DAG Cards.

Card Configuration

Attribute Reference

Before you can change a card's configuration you must:

- obtain a reference to the card,
- then reference the desired component, and
- reference the component's attribute that you wish to change.

Once you have the attribute reference you can use it to retrieve and modify the attribute value.

For example, to see if a particular port is active, you would first obtain a reference to the card, then a reference to the port component, and finally a reference to the port component `active.`

Attribute Value

Reading the value returned by the attribute reference provides information about the port status. Writing a value to the attribute reference would configure the port status.

A sample program that displays the `"active"` attribute for all parts on a DAG card is shown in Example Program (see page 5)

Attribute Type

There are two types of attributes associated with components on DAG cards.

- Status attributes: used to represent status and statistics information, and
- Configuration attributes: used to represent configuration information.

You can use the `dag_config_get_attribute_config_status` function to check if an attribute is marked as a status or configuration attribute.

Configuration Attribute

Configuration attributes represent properties of the card that can be modified. They include such items as:

- POS or ATM mode for SONET cards
- Auto-negotiation mode on/off for Ethernet cards
- Variable or fixed-length packet capture
- Snap length for packet capture
- Amount of memory allocated to each receive and transmit stream

Status Attribute

Status attributes represent the card properties that are read-only and can not be modified. They include such items as:

- Physical layer error indicators.
- PCI bus speed.
- Number of frames that failed the Frame Checksum.
- Number of receive and transmit streams supported by the firmware.

Note: The precise set of attributes and components presented by the API depends on the model of DAG card and the capabilities of the loaded firmware image(s).
The API provides functions to deal with attributes depending on whether they are configuration or status specific.

Example Program

The following program illustrates how the Configuration & Status API is used. It queries the `active` attributes of all ports on the card and displays the result. For the sake of clarity the error-handling code has been omitted from this example.

```
#include "dag_component.h"

#include <stdio.h>
#include <stdlib.h>

int
main(int argc, const char* argv[])
{
   dag_card_ref_t card_ref = NULL;
   dag_component_t root_component = NULL;
   uint32_t count;
   uint32_t i;

   /* Get a reference to the card. */
   card_ref = dag_config_init("/dev/dag0");

   /* Get a reference to the root component. */
   root_component = dag_config_get_root_component(card_ref);

   /* Find out how many ports the card has. */
   count = dag_component_get_subcomponent_count_of_type(root_component,
   kComponentPort);

   for (i = 0; i < count; i++)
   {
          dag_component_t port = NULL;
          attr_uuid_t active_uuid = 0;
          uint8_t val = 0;

          /* Get a reference to the port. */
          port = dag_component_get_subcomponent(root_component,
   kComponentPort, i);

          /* Get a reference to the active attribute of the port. */
          active_uuid = dag_component_get_config_attribute_uuid(port,
   kBooleanAttributeActive);

          /* Read and display the value of the attribute. */
          val = (uint8_t) dag_config_get_boolean_attribute(card_ref,
   active_uuid);
                 printf("Port %u active = %u\n", i, val);
   }
   /* Dispose of the card. */
   dag_config_dispose(card_ref);

   return EXIT_SUCCESS;
}
```

Using the API

Pre-requisites

To use the Configuration & Status API effectively you will need to have some software development experience. This Programming Guide assumes that you are competent at programming in C and are familiar with the Linux operating systems and the distribution you have installed.

Header Files

Whenever you use the Configuration & Status API you must always include the following header files:

- **`dag_config.h`**
  Contains routines that relate to the card as a whole e.g. getting an initial reference to the card, loading firmware, finding a component by name, as well as routines that retrieve and set values on attributes.
- **`dag_component.h`**
  Contains routines that operate on components, e.g. getting the root component, getting subcomponents, getting attributes of a component.
- **`dag_component_codes.h`**
  Contains the codes e.g. `kComponentStream` used to refer to components.
- **`dag_attribute_codes.h`**
  Contains the codes e.g. `kBooleanAttributeVarlen` used to refer to attributes and enumerated types for attributes that have a restricted range of valid values.

Alternatively you may use the files **`dag_config_api.h`**. This is provided simply for convenience as its only function is to include the four essential files listed above.

FreeBSD/Linux

If you are running a FreeBSD or Linux operating system the header files are installed in `/usr/local/include` by default. However if you want to change this location you can do so when running the `configure` script.

Library files are installed in `/usr/local/lib` by default. You can also change this locations when running the `configure` script.

Windows®

If you are running a Windows® operating system the header files are installed in `%Program Files%\Endace\dag-x.y.z\include.` Stub library files are installed in `%Program Files%\Endace\dag-x.y.z \lib\windows\VCproject\Release` and Runtime library files are installed in `%System%`.

Note: The phrases in `%%` are standard system locations and may vary from machine to machine.

Chapter 3: Components and Attributes

Overview

The model of a DAG card implied by the API is a hierarchical tree of *components*. The top component in the tree is called the *root* component which contains a reference to the attached DAG card and subcomponents.

Each subcomponent of the root has a set of attributes associated with it which defines the configuration of the module at any point in time. Changing the value of the component attributes directly changes the behavior of the corresponding modules.

Note: Not all components and attributes are common to all DAG Cards. Some components and attributes listed below may have changed.

Your Card's Components and Attributes

To obtain a list of your DAG card's components and attributes, run this command at a prompt:

```
dagconfig -T -v2
```

Component Definitions

The different types of components associated with DAG cards are shown below:

KComponent	Description
Card Info	Card information.
Connection setup	For the 3.7T represents a connection component. Allows connections to be added and removed.
Counter	The counter component.
Demapper	Demapper components are used to provide a higher level of functionality over the base framer.
Drop	The count of dropped packets.
DSM	DSM module information
DUCK	The DUCK (DAG Universal Clock Kit). Used to configure the time keeping abilities of the DAG card.
E1T1	Represents the E1T1 deframer/framer.
ERF MUX	ERF MUX Component. This component can be used to merge or split the receive streams on the card.
Framer	Represents the framer component. A Framer encapsulates data within a SONET Frame for transmit.
General	The General register component.
GPP	The Generic Packet Processor captures the packet. It can be configured using the `snaplen` attribute to define a fixed number of bytes to capture from each packet.
Hardware Monitor	The hardware monitor (temperature, fan, voltage etc.)
Interface	The Counter Statistic Interface component.
IPF	Component for IPF information
LED Controller	Represents the LED controller for the pod.
Mapper	Mapper components are used to provide a higher level of functionality over the base framer.
MiniMac Statistics	Represents the statistics module for each port.
Mux	The mux component can be used to merge or split the receive streams on the card.
Optics	Represents the optics component on the card.
Packet Capture	The packet capture statistics module.
PBM	The PCI Burst Manager handles the transfer of captured packets to the receive memory stream and from the transmit stream back to the card for transmitting. This component can be used to check the size of the memory buffer allocated, and to count the number of transmit and receive streams present. On some DAG cards you can set the overlap attribute to enable inline forwarding of packets.
PCS	The Physical Coding Sublayer (PCS) component.
Phy	Represents the physical layer on a card.
Port	Used to configure and read attributes specific to the line. These differ widely between cards, although some are common depending on the protocol the card is designed for. For example, all Ethernet cards have similar line attributes, however, a SONET card will not have many line attributes in common with an Ethernet card.
Root	A special component that has no attributes. All other components are children of the root component.
SC256	Represents the optional coprocessor when present on the card.
SONET PP	SONET Packet processor component on the new cards.
Sonic	Controls attributes of the SONET/SDH deframer.
SRGPP	**Depreciated** please use `kComponetGpp`.
Steering	The steering component. This allows one to choose a algorithm to steer the received packets. The steering algorithm allows the packets to be directed to different memory holes depending on for example a crc hash function. See also: Load Balancing Steering Options.
Stream	Represents either a receive or transmit stream and can be used to allocate memory to the stream. The number of stream components differs depending on the firmware

KComponent	Description
	image.
Stream Features	The stream component models a receive stream or transmit stream. The number of streams depends on the loaded firmware image. This component can be used to allocate memory to the stream it represents.
Terf	Represents the terf register on cards that have the appropriate firmware loaded.
XGMII	The XGMII component.

Components per card

DAG cards 3.7D to 5.0SG2A

Component	3.7D	3.7G	3.7T	3.8S	4.3S	4.3GE	4.5G2/G4	5.0SG2 / 5.0SG2A
`kComponentCardInfo`	1	1	1	1	1	1	1	1
`kComponent ConnectionSetup`			1					
`kComponentCounter`								
`kComponentDemapper (Channelized)`								
`kComponentDemapper (Concatenated)`			1					
`kComponentDrop`							2	8
`kComponentDSM`								
`kComponentDUCK`	1	1	1	1	1	1	1	1
`kComponentE1T1`								
`kComponentErfMux`								
`kComponentFramer`			1					
`kComponentGeneral`				1		1	1	1
`kComponentGpp`	2	2		2	1	1	1	1
`kComponentHardware Monitor`							1	1
`kComponentInterface`								1
`kComponentIPF`								1
`kComponentLED Controller`			1					
`kComponentMapper (Channelized)`								
`kComponentMapper (Concatenated)`								
`kComponentMiniMac Statistics`							2	
`kComponentMux`		1						
`kComponentOptics`								2
`kComponentPbm`	1	1	1	1	1	1	1	1
`kComponentPCS`								
`kComponentPhy`								2
`kComponentPort`	2	2	16	2	1	2	2/4	2
`kComponentSC256`	1	1				1		
`kComponentSonetPP`								2
`kComponentSonic`								
`kComponentSteering`							1	
`kComponentStream`	3	3	1 or 2	2	2	2	4	4/16
`kComponentStream Features`								
`kComponentTerf`		1		1	1	1	1	1
`kComponentXGMII`								

DAG cards 5.2SXA to 8.4I

Component	5.2SXA	5.2X	5.4 / 5.4A	6.2SE	7.1S	8.1SX	8.1X	8.2X	8.4I
`kComponentCardInfo`	1	1	1	1	1	1	1	1	1
`kComponent ConnectionSetup`									
`kComponentCounter`									
`kComponentDemapper (Channelized)`									
`kComponentDemapper (Concatenated)`					4				
`kComponentDrop`	1	2							
`kComponentDSM`		1	1					1	
`kComponentDUCK`	1	1		1	1	1	1	1	
`kComponentE1T1`					4				
`kComponentErfMux`					1				
`kComponentFramer`									
`kComponentGeneral`	1	1	1			1	1	1	1
`kComponentGpp`	1	1	1	1	1	1	1	1	2
`kComponentHardware Monitor`	1	1	1		1	1	1	1	1
`kComponentInterface`	1		1			1	1		1
`kComponentIPF`	1		1						
`kComponentLED Controller`									
`kComponentMapper (Channelized)`					4				
`kComponentMapper (Concatenated)`					4				
`kComponentMiniMac Statistics`									
`kComponentMux`									
`kComponentOptics`	1		2		4	1	1		
`kComponentPbm`	1	1	1	1	1	1	1	1	1
`kComponentPCS`	1					1	1		
`kComponentPhy`			2		1				
`kComponentPort`	1	1	2	1	4	1	1	2	
`kComponentSC256`									
`kComponentSonetPP`	1		2						
`kComponentSonic`					4				
`kComponentSteering`		1	1					1	
`kComponentStream`	2	4	2/4		1/2	2	2	2	2
`kComponentStream Features`									
`kComponentTerf`	1	1	1	1	1	1		1	
`kComponentXGMII`	1	1				1	1	1	

kComponentCardInfo

Card Information.

3.7D, 3.7G, 3.7T, 3.8S, 4.3GE, 4.3S, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.2SXA, 5.2X, 5.4S-12, 5.4SG-48, 5.4GA, 5.4SA-12, 5.4SGA-48, 7.1S, 8.1SX, 8.1X, 8.2X, 8.4I

Attribute	Description	Access
`kStringAttributeFactoryFirmware`	Indicates the Factory firmware version.	status
`kStringAttributePciInfo`	Physical slot information.	status
`kStringAttributeUserFirmware`	Indicates the User firmware version.	status
`kUint32AttributeActiveFirmware`	Indicates which partition is the current Active Firmware.	status
`kUint32AttributeSerialID`	Indicates the DAG Card's Serial ID.	status
`kUintAttributeCopro`	Indicates if a Coprocessor is fitted and the type.	status

5.0SG2A, 5.2SXA, 5.4GA, 5.4SA-12, 5.4SGA-48

Attribute	Description	Access
`kStringAttributeUserCoproFirmware`	Indicates the User Co-Processor firmware version.	status
`kStringAttributeFactoryCoproFirmware`	Indicates the Factory Co-Processor firmware version.	status

kComponentConnectionSetup

Represents a Connection component, allows connections to be added and removed.

3.7T

Attribute	Description	Access
`kNullAttributeClearConnections`	Deletes all connections.	config
`kStructAttributeAddConnection`	Adds a single connection to the card. Connection information must be specified for the connection. Refer to `connection_description_37t_t` for more information.	config
`kUint32AttributeDeleteConnection`	Deletes a single connection. Requires the connections number of the connection to be deleted.	config
`kUint32AttributeGetLastConnectionNumber`	Retrieves the connection number of the last connection created. This number is only valid for the last connection created in this instance of the library. It is recommended to use the connection number supplied by the system with `kStructAttributeAddConnection`.	status

kComponentCounter

The Counter component.

5.4S-12,

Attribute	Description	Access
`kBooleanAttributeAccess`	Access type (Direct -0- or Indirect -1-)	
`kBooleanAttributeLatchClear`	Counter Latch & Clear	
`kBooleanAttributeValueType`	Counter value type (address or value)	
`kUint32AttributeCounterID`	Counter type ID.	
`kUint32AttributeCounterSize`	Counter size.	
`kUint32AttributeSubFunction`	Counter Sub-function.	
`kUint64AttributeValue`	Value of counter.	

kComponentDemapper (channelized)

The demapper for the channelized SONET.

7.1S

Attribute	Description	Access
`kBooleanAttributeTimeStampEnd`	Use to change where in the packet the time stamp is measured from.	config
`kNullAttributeClearConnections`	Use this attribute to clear the all connections on the card.	config
`kStructAttributeAddConnection`	Use to add a connection.	config
`kUint32Attribute71sChannelized RevisionID`	Use to retrieve revision ID of the ATM/HDLC demapper. The revision ID can be used to determine the features the demapper supports.	status
`kUint32AttributeDeleteConnection`		config
`kUint32AttributeGetLastConnection Number`	Use to get the connection number of the last connection added.	status
`kUint32AttributeHDLCSnaplength`	Set the snaplength for incoming hdlc packets	config
`kUint32AttributeRAWSnaplength`	Set the snaplength for incoming raw packets	config
`kUint32AttributeSonetType`	Indicates whether the component is configured for channelized or concatenated	status

KComponentDemapper (concatenated)

The demapper for the concatenated SONET.

7.1S

Attribute	Description	Access
`kBooleanAttributeIdleCellMode`	Set this to pass idle cells.	config
`kBooleanAttributeLossOfCellDelineation`	Indicates if the demapper is in LCD (loss of cell delineation) mode.	status
`kBooleanAttributePayloadScramble`	Enable/disables ATM cell or PoS frame scrambling.	config
`kUint32AttributeCrcSelect`	Selects CRC16, 32 or to turn off CRC checking.	config
`kUint32AttributeHECCount`	Count of the number of cells with HEC errors.	status
`kUint32AttributeIdleCellCount`	Count of the number of idle cells since this attribute was last read.	status
`kUint32AttributeNetworkMode`	Switches between POS or ATM modes.	config
`kUint32AttributeSonetType`	Indicates if the component is configured for channelized or concatenated.	status

kComponentDrop

Present on modules with a drop counter. Hash load balancing firmware supports this component.

4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.2SXA, 5.2X, 5.4S-12, 5.4SG-48, 5.4SGA-48, 8.2X,

Attribute	Description	Access
`kUint32AttributeStreamDropCount`	Packets dropped by this stream	status

kComponentDSM

DSM module information.
5.0SG2, 5.2X, 5.4S-12, 5.4SG-48, 8.2X

Attribute	Description	Access
`kBooleanAttributeDSMBypass`	To enable/disable DSM functionality	config

kComponentDUCK

The DUCK (DAG Universal Clock Kit). Used to configure the time keeping abilities of the DAG card.

Attribute	Description	Access
`kBooleanAttributeDUCKAuxInput`	DAG auxiliary synchronization input enabled.	config
`kBooleanAttributeDUCKHostInput`	Host generated synchronization input enabled. (not used)	config
`kBooleanAttributeDUCKHostOutput`	DAG synchronization output source is from Host PC.	config
`kBooleanAttributeDUCKLoop`	DAG synchronization output source is selected DUCK input signal.	config
`kBooleanAttributeDUCKOverInput`	DAG internal synchronization input enabled.	config
`kBooleanAttributeDUCKOverOutput`	DAG synchronization output source is internal DUCK clock.	config
`kBooleanAttributeDUCKRS422Input`	External RS422 synchronization input enabled.	config
`kBooleanAttributeDUCKRS422Output`	DAG synchronization output source is RS422 signal.	config
`kBooleanAttributeDUCKSynchronized`	Indicates if the DUCK is synchronized.	status
`kInt32AttributeDUCKFrequencyError`	DUCK clock frequency error in parts per billion (ppb).	status
`kInt32AttributeDUCKPhaseError`	DUCK phase error (offset) measured in nanoseconds (ns).	status
`kNullAttributeDUCKClearStats`	Clear the duck statistics.	config
`kNullAttributeDUCKSetToHost`	Set the DAG clock to the host PC clock.	config
`kNullAttributeDUCKSync`	Try to synchronize the DUCK	config
`kStructAttributeDUCKTimeInfo`	Holds host time values for when DUCK statistics were last cleared and the last synchronization event.	Status
`kUint32AttributeDUCKCrystalFrequency`	Estimated crystal oscillator frequency in Hz.	Status
`kUint32AttributeDUCKFailures`	Increments by one when `kUint32 AttributeDUCKPhaseError > kUint32 AttributeDUCKThreshold` for 10 consecutive seconds.	Status
`kUint32AttributeDUCKLongestPulse Missing`	Longest contiguous sequence of synchronization signals missing.	Status
`kUint32AttributeDUCKPulses`	Count of synchronization signals received.	Status
`kUint32AttributeDUCKResyncs`	Increments by one when `kUint32Attribute DUCKPhaseError` ≥ 1 second causing the DUCK to reset to factory defaults, except for the input selection, and attempt to resynchronize.	status
`kUint32AttributeDUCKSinglePulses Missing`	Count of times when a single expected synchronization signal is missing.	status
`kUint32AttributeDUCKSyncTimeout`	The timeout value in seconds before the synchronization operation fails.	config
`kUint32AttributeDUCKSynthFrequency`	Target DUCK operating frequency.	status
`kUint32AttributeDUCKThreshold`	DUCK is synchronized when `kUint32 AttributeDUCKPhaseError` < `kUint32 AttributeDUCKThreshold.`	config
`kUint32AttributeDUCKWorstFrequency Error`	Highest value of `kUint32AttributeDUCK FrequencyError` since last DUCK reset or DUCK statistics reset.	status
`kUint32AttributeDUCKWorstPhaseError`	Highest value of `kUint32AttributeDUCK PhaseError` since last DUCK or DUCK statistics reset.	status

kComponentE1T1

Represents the E1T1 deframer/framer

7.1S

Attribute	Description	Access
`kBooleanAttributeE1T1GenerateAlarmIndication`	Sets the alarm indication to OFF or ON.	config
`kBooleanAttributeE1T1LinkFramingError`	Indicates the stream's framing experienced an error since last selected.	status
`kBooleanAttributeE1T1LinkSynchronized`	Used to check if the stream number is synchronized to the framing information.	status
`kBooleanAttributeE1T1LinkSynchronizedDown`	Indicates the stream has lost framing lock since last selected.	status
`kBooleanAttributeE1T1LinkSynchronizedUp`	Indicates the stream has synchronized to the framing information since last selected.	status
`kBooleanAttributeE1T1LinkAIS`	Indicates the stream was in AIS mode since last selected.	status
`kBooleanAttributeE1T1LinkCRCError`	Indicates the stream has seen CRC error since last selected. Only valid if E1 with CRC is selected for that stream.	status
`kUint32AttributeE1T1StreamNumber`	Sets the stream number to read the status data from.	config
`kUint32AttributeLineType`	Set the line type. For valid values	config

kComponentErfMux

ERF MUX Component. This component can be used to merge or split the receive streams on the card.

7.1S

Attribute	Description	Access
`kUint32AttributeLineSteeringMode`	Steering direction for packets received from the line	config
`kUint32AttributeIXPSteeringMode`	Steering direction for packets received from the IXP	config
`kUint32AttributeHostSteeringMode`	Steering direction for packets received from the host	config

kComponentFramer

Represents the framer component. A Framer encapsulates data within a SONET Frame for transmit

3.7T

Attribute	Description	Access
`kBooleanAttributeClear`	Use to clear the framer. This will clear the statistics counters in the framer.	config
`kBooleanAttributeReset`	Resets the framer.	config

kComponentGeneral

The General register component.

4.3GE,

Attribute	Description	Access
`kBooleanAttributeSuppressError`	Suppress most of error	config

kComponentGpp

The generic packet processor component captures the packet. It can be told to capture, using the `snaplen` attribute, a fixed number of bytes from the wire.

3.7D, 3.7G, 3.8S, 4.3GE, 4.3S, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.2SXA, 5.2X, 5.4S-12, 5.4SG-48,5.4GA, 5.4SGA-12, 5.4SGA-48, 7.1S, 8.1SX, 8.1X, 8.2X, 8.4I

Attribute	Description	Access
`kBooleanAttributeAlign64`	Turns 64-bit alignment ON/OFF. If on the ERF records captured will be 64 bit aligned.	Config
`kBooleanAttributeVarlen`	Variable length capture. If disabled the record is padded up to the number of bytes specified by the snap length attribute.	Config
`kUint32AttributeDropCount`	A count of the packets dropped on each port. One per port on the DAG card.	Status
`kUint32AttributeInterfaceCount`	The number of interfaces in the card.	Status
`kUint32AttributeSnaplength`	Sets the number of bytes to capture per packet.	Config
`kBooleanAttributeActive`	Enables or disables the port. One per port on the DAG card.	Config

kComponentHardwareMonitor

The hardware monitor (temperature, fan, voltage etc..)

4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.2SXA, 5.2X, 5.4S-12, 5.4SG-48, 5.4GA, 5.4SA-12, 5.4SGA-48, 7.1S, 8.1SX, 8.1X, 8.2X, 8.4I

Attribute	Description	Access
`kUint32AttributeTemperature`	Indicates the current temperature value.	status
`kUint32AttributeVoltage`	Indicates the current voltage.	status

4.5G4, 5.0SG2, 5.0SG2A, 5.2SXA, 5.2X, 5.4S-12, 5.4SG-48, 5.4GA, 5.4SA-12, 5.4SGA-48, 7.1S, 8.1SX, 8.1X, 8.2X, 8.4I

Attribute	Description	Access
`kFloatAttributeVoltage`	The measured voltage level line.	status

kComponentInterface

The Counter Statistic Interface component.

5.0SG2, 5.0SG2A, 5.2SXA, 5.4S-12, 5.4SG-48, 5.4SGA-48, 8.1SX, 8.1X, 8.4I

Attribute	Description	Access
`kBooleanAttributeLatchClear`	Latch & Clear set up.	config
`kUint32AttributeCounterDescBaseAdd`	Counter description base address.	status
`kUint32AttributeCounterValueBaseAdd`	Counter value base address	status
`kUint32AttributeCSIType`	Counter Statistics Interface type.	status
`kUint32AttributeNbCounters`	Number of counters in the Counter Statistics Interface.	status

kComponentIPF

Component for IPF information

5.0SG2A, 5.2SXA, 5.4SGA-48,

Attribute	Description	Access
`kBooleanAttributeIPFDropEnable`	Indicates whether the IPF will drop packets that are supposed to go to none of the streams.	config
`kBooleanAttributeIPFEnable`	To enable/disable IPF functionality.	config
`kBooleanAttributeIPFLinkType`	Indicates whether the link type is Ethernet or PoS/PPP.	status
`kBooleanAttributeIPFRulesetInterface0`	Set the rule to interface 0 - can be used for hot swapping.	config
`kBooleanAttributeIPFRulesetInterface1`	Set the rule to interface 1 - can be used for hot swapping.	config
`kBooleanAttributeIPFSelLctr`	Indicates whether the color is embedded within the loss counter field.	config
`kBooleanAttributeIPFShiftColour`	Indicates whether the higher or lower 2 bits of the color is dropped when hash mode is enabled.	config
`kBooleanAttributeIPFUseRXError`	Indicates whether RX error is used to show the pass/drop status.	config

kComponentLEDController

Represents the LED controller for the pod

3.7T

Attribute	Description	Access
`kUint32AttributeLEDStatus`	Sets the status of an LED. There are 32 of these attributes in this component that represent the 32 LEDs on the pod. Each LED can be assigned a status. For valid values.	config
`kUint32AttributePeriod`	Sets the frequency with which the LED will blink in 100ths of a second. Note you cannot assign a different frequency to each LED because of hardware limitations of the PCA9552 chip This means one frequency is assigned to all LEDs so, depending on their status, all will blink at that frequency.	config
`kUint32AttributeDutyCycle`	Set the period of time LED off during blinking.	config

kComponentMapper (3.7T)

Represents a demapper component. Demapper components are used to provide a higher level of functionality over the base framer.

3.7T

Attribute	Description	Access
`kUint32AttributeLossOfCell DelineationCount`	Indicates the number of LCD instances.	config
`kUint32AttributeDropCount`	Count of the number of packets dropped.	status
`kStructAttributeErfMux`	Set the Erf Mux (steering) on the 3.7T card.	config
`kBooleanAttributeCounterLatch`	'Latch and Clear' the receive/transmit statistics counters.	config
`kBooleanAttributeTimeStampEnd`	Indicates when the timestamp is to be added to the record.	config

kComponentMapper (channelized)

Represents a mapper component. Mapper components are used to provide a higher level of functionality over the base framer.

7.1S

Attribute	Description	Access
`kNullAttributeClearConnections`	To delete all connections	config
`kStructAttributeAddConnection`	To add a connection	config
`kUint32Attribute71sChannelized RevisionID`	revision ID of the frames	status
`kUint32AttributeDeleteConnection`	Delete a single connection	config
`kUint32AttributeGetLastConnection Number`	Connection number last created.	status
`kUint32AttributeSonetType`	Indicates if the component is configured for channelized or concatenated.	status

kComponentMapper (concatenated)

Represents a mapper component. Mapper components are used to provide a higher level of functionality over the base framer.

7.1S

Attribute	Description	Access
`kBooleanAttributePayloadScramble`	Enable/disable ATM cell or PoS frame scrambling.	config
`kUint32AttributeCrcSelect`	`Use to select CRC16, 32 or to turn off CRC checking.`	config
`kUint32AttributeNetworkMode`	Sets the port to POS/ATM mode.	config
`kUint32AttributeSonetType`	Indicates if the component is configured for channelized or concatenated.	status

kComponentMiniMacStatistics

Represents the statistics module for each port. Firmware dependent

Attribute	Description	Access
`kBooleanAttributeCounterLatch`	Set this attribute before reading statistics. It latches the statistics counters so they can be read in a consistent state.	config
`kBooleanAttributeCrcErrorEverHi`	Indicates if a CRC error was seen since last read.	status
`kBooleanAttributeCrcErrorEverLo`	Indicates if a CRC error was set to 0 since last read.	status
`kBooleanAttributeLinkCurrent`	Indicates if there is a current Link error.	status
`kBooleanAttributeLinkEverHi`	Indicates if a Link error was seen since last read.	status
`kBooleanAttributeLinkEverLo`	Indicates if a Link error was set to 0 since last read.	status
`kBooleanAttributeLossOfFramingCurrent`	Indicates if there is a current Loss of Framing error.	status
`kBooleanAttributeLossOFFramingEverHi`	Indicates if a Loss of Framing error was seen since last read.	status
`kBooleanAttributeLossOfFramingEverLo`	Indicates if a Loss of Framing error was set to 0 since last read.	status
`kBooleanAttributeLossOfSignalCurrent`	Indicates if a there is a current LOS.	status
`kBooleanAttributeLossOfSignalEverHi`	Indicates if a Loss Of Signal error was seen since last read.	status
`kBooleanAttributeLossOfSignalEverLo`	Indicates if a Loss of Signal error was set to 0 since last read.	status
`kBooleanAttributeMiniMacLostSync`	Indicates if the Mini Mac has lost Synchronization.	status
`kBooleanAttributePeerLinkCurrent`	Indicates if there is a current Peer Link error.	status
`kBooleanAttributePeerLinkEverHi`	Indicates if there ever was a Peer Link error since last read.	status
`kBooleanAttributePeerLinkEverLo`	Indicates if there ever was a Peer Link error set to 0 since last read.	status
`kBooleanAttributeRefreshCache`	It is necessary to cache the statistics values before reading them as they are cleared when any of the values are read from the component.	config
`kBooleanAttributeRemoteErrorCurrent`	Indicates if there is a current Remote Error.	status
`kBooleanAttributeRemoteErrorEverHi`	Indicates if a there was a Remote Error since last read .	status
`kBooleanAttributeRemoteErrorEverLo`	Indicates if there was a Remote error set to 0 since last read.	status
`kBooleanAttributeSFPTxFaultCurrent`	Indicates if there is a current SFP Tx Fault.	status
`kBooleanAttributeSFPTxFaultEverHi`	Indicates if there was a SFP Tx Fault since last read.	status
`kBooleanAttributeSFPTxFaultEverLo`	Indicates if there was a SFP Tx Fault set to 0 since last read.	status
`kUint32AttributeBadSymbols`	Indicates the number of bad symbols since last read.	status
`kUint32AttributeConfigSequences`	Indicates the number of configuration Sequences since last read.	status
`kUint32AttributeCrcErrors`	Indicates the number of CRC Errors since last read.	status
`kUint32AttributeRemoteErrors`	Indicates the number of remote errors since last read.	status
`kUint32AttributeRxFrames`	Indicates the number of RX Frames since last read.	status
`kUint32AttributeTxFrames`	Indicates the number of TX Frames since last read.	status
`kuint64AttributeRxBytes`	Indicates the number of RX bytes since last read.	status
`kuint64AttributeTxBytes`	Indicates the number of TX bytes since last read.	status

kComponentMux

Represents the mux component. This component can be used to merge or split the receive streams on the card.

3.7G

Attribute	Description	Access
`kUint32AttributeSteer`	The method to use to steer the incoming packet	Config
`kBooleanAttributeSwap`	Disable or enable tx interface swapping on the card in the ERF header	config

kComponentOptics

Represents the optics component on the card.

5.0SG2, 5.0SG2A, 5.2SXA, 5.4S-12, 5.4SG-48, 5.4SGA-48, 7.1S, 8.1SX, 8.1X,

Attribute	Description	Access
`kBooleanAttributeDetect`	Indicates the presence of the XFP module.	status
`kBooleanAttributeLaser`	Indicates the status of the optics transmit laser. Allows the laser to be turned On and OFF.	config

5.0SG2, 5.0SG2A, 5.2SXA, 5.4S-12, 5.4SG-48, 5.4SGA-48, 8.1SX, 8.1X,

Attribute	Description	Access
`kBooleanAttributeLossOfSignal`	Indicates XFP loss of signal.	status

5.0SG2A, 5.2SXA, 5.4S-12, 5.4SG-48, 5.4SGA-48,

Attribute	Description	Access
`kBooleanAttributeTransceiverMonitoring`	Indicates diagnostic monitoring has been implemented	
`kStringAttributeTransceiverLinkLength`	Fiber Channel link length	
`kStringAttributeTransceiverMedia`	Fiber Channel media	
`kStringAttributeTransceiverVendorName`	Vendor name of the module	
`kStringAttributeTransceiverVendorPN`	Vender Part number.	
`kUint32AttributeTransceiver ExtendedIdentifier`	Extended Module type identifier	
`kUint32AttributeTransceiverIdentifier`	Module type identifier	

5.2SXA, 7.1S, 8.1SX, 8.1X,

Attribute	Description	Access
`kBooleanAttributeSFPPwr`	Indicates SFP power.	status

7.1S,

Attribute	Description	Access
`kBooleanAttributeSignal`	Indicates if the optics are detecting input signal.	status

kComponentPbm

The PCI Burst Manager component handles the transfer of captured packets to the receive memory stream and from the transmit stream back to the card for transmitting. This component can be used to check the size of the memory buffer allocated, and to count the number of transmit and receive streams present. On some cards one can set the overlap attribute to enable inline forwarding of packets.

3.7D, 3.7G, 3.8S, 4.3GE, 4.3S, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.2SXA, 5.2X, 5.4S-12, 5.4SG-48, 5.4GA, 5.4SA-12, 5.4SGA-48, 7.1S, 8.1SX, 8.1X, 8.2X, 8.4I

Attribute	Description	Access
`kBooleanAttributeDrop`	If ON packets are dropped at the individual stream that has filled up. If OFF packets are dropped at the GPP.	config
`kBooleanAttributeOverlap`	Shares the memory hole between the receive and transmit streams to support inline forwarding.	config
`kUint32AttributeBufferSize`	The size of the buffer allocated to the DAG card.	status
`kUint32AttributePCIBusSpeed`	A number representing the PCI bus speed. See `pci_bus_speed_t`.	status
`kUint32AttributeRxStreamCount`	Count of the number of receive streams.	status
`kUint32AttributeTxStreamCount`	Count of the number of transmit streams.	status

kComponentPCS

The Physical Coding Sublayer (PCS) component

5.2SXA, 8.1SX, 8.1X,

Attribute	Description	Access
`kBooleanAttributeHiBER`	High Bite Error Rate	status
`kBooleanAttributeLink`	Receive link status	status
`kBooleanAttributeLock`	Signal lock if 1	status
`kBooleanAttributeReset`	Reset	config
`kBooleanAttributeRxFault`	Receive signal fault	status
`kBooleanAttributeRxFIFOOverflow`	Received FIFO overflow	status
`kBooleanAttributeTxFault`	Transmit signal fault	status
`kBooleanAttributeTxFIFOOverflow`	Transmit FIFO overflow	status
`kUint32AttributeBERCounter`	Bit Error Rate counter	status
`kUint32AttributeErrorBlockCounter`	Block error counter.	status

kComponentPhy

Represents the physical layer on a card.

5.0SG2, 5.0SG2A, 5.4S-12, 5.4SG-48, 5.4SGA-48

Attribute	Description	Access
`kBooleanAttributeActive`	Enables/disabled the link or port.	config
`kBooleanAttributeDiscardData`	Configures the criteria for discarding data.	config
`kBooleanAttributeLock`	Indicates the card is locked on its reference clock.	status
`kBooleanAttributeLossOfSignal`	Indicates the card has experienced loss of signal.	status
`kBooleanAttributePhyBistEnable`	Enables the built in self test (BIST).	config
`kBooleanAttributePhyKillRxClock`	Reassert to start receiving.	config
`kBooleanAttributePhyTxClockOff`	Clock off transmit.	config
`kBooleanAttributeReset`	Resets the component.	config
`kBooleanAttributeTransmitLockError`	Indicates a transmit lock error.	status.
`kUint32AttributeConfig`	Configures the main clock.	config
`kUint32AttributeMasterSlave`	Set the SONET clock master/slave status. For valid values see `master_slave_t`.	config
`kUint32AttributePhyRate`	Selects the Phy rate.	config
`kUint32AttributePhyRefSelect`	Selects the Phy reference.	config

5.0SG2, 5.0SG2A, 5.4S-12, 5.4SG-48, 5.4SGA-48, 7.1S

Attribute	Description	Access
`kBooleanAttributeEquipmentLoopback`	Enables/disables equipment loopback.	config
`kBooleanAttributeFacilityLoopback`	Enables/disables facility loopback.	config

kComponentPort

The port component is generally used to configure and read attributes specific to the line. The specific attributes differ widely between cards. However, there is some commonality depending on the protocol for which the card is designed. For example, all Ethernet cards have similar attributes associated with their port component. However, a SONET card port component will not share many attributes in common with an Ethernet card's port component.

Attribute	Description	Access	DAG Card
`kBooleanAttributeActive`	Enables/disables the port or link	config	3.8S, 4.3GE, 4.3S, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.4SG48, 5.4GA, 5.4SGA48,
`kBooleanAttributeAICM23Cbit`	Application Identification Channel (AIC) Cbit/M23 mode.	status	3.7D,
`kBooleanAttributeAlarmIndicationSignal`	Indicates when the receive frame processor has in Alarm Indication Signal.	status	3.7D,
`kBooleanAttributeAlarmSignal`	Indicates if link is experiencing AIS.	status	3.7T, 3.8S,
`kBooleanAttributeATMScramble`	Sets scrambling on the port, only when firmware supports ATM receive. Can be true (ON) or false (OFF)	config	3.7T,
`kBooleanAttributeAutonegotiationComplete`	This attribute is only valid if `kBooleanAttributeNic` is enabled. In this mode this attribute indicates if Ethernet auto-negotiation has completed.	status	3.7G, 4.3GE,
`kBooleanAttributeB1Error`	Indicates if the link is experiencing B1 errors.	status	3.8S,
`kBooleanAttributeB2Error`	Indicates if the link is experiencing B2 errors.	status	3.8S,
`kBooleanAttributeB3Error`	Indicates if the link is experiencing B3 errors.	status	3.8S,
`kBooleanAttributeByteCount`	Sets the maximum number of bytes allowed per packet.	config	
`kBooleanAttributeCore`	Indicates if the core is ON or OFF.	config	7.1S,
`kBooleanAttributeCounterLatch`	Used to latch the counter attributes on the card to allow them to be read. This must be set to 1 before reading values from the following: • `kUint64AttributeBadSymbol` • `kUint64AttributeCrcFailkUint64AttributeInternalMACError` • `kUint64AttributeRxBytes` • `kUint64AttributeRxFrames` • `kUint64AttributeTransmitSystemError` • `kUint64AttributeTxBytes` • `kUint64AttributeTxFrames`	config	3.7D, 4.3GE, 4.3S,
`kBooleanAttributeCrcStrip`	Enables/disables CRC stripping from received packets.	config	4.3S,
`kBooleanAttributeDataOutOfLock`	Indicates a Data Out Of Lock error condition.	config	4.3S,
`kBooleanAttributeDescramble`	Enable or disable SONET frame scrambling.	config	3.7D
`kBooleanAttributeDriverMonitorOutput`	Indicates when a transmit driver failure is detected.	status	3.7T
`kBooleanAttributeE1T1AISError`	Indicates if there was an Alarm Indication Signal Error	status	3.7T
`kBooleanAttributeE1T1CRCError`	Indicates if there was a CRC error.	status	3.7T
`kBooleanAttributeE1T1Framer`	Indicates if there was a framer error.	status	
`kBooleanAttributeE1T1FramerError`	Indicates if there was a framer error.	status	3.7T
`kBooleanAttributeE1T1Link`	Indicates if the link is up and running.	status	3.7T
`kBooleanAttributeE1T1Rx0`	If this is set it mean that nothing is being processed by the SONIC E1/T1 framer. This is caused by faulty hardware.	status	3.7T

Attribute	Description	Access	DAG Card
`kBooleanAttributeE1T1Rx1`	If this is set it means that nothing is being processed by the SONIC E1/T1 framer. This is caused by faulty hardware.	status	3.7T
`kBooleanAttributeE1T1Tx0`	If this is set it means that nothing is being processed by the SONIC E1/T1 framer. This is caused by faulty hardware.	status	3.7T
`kBooleanAttributeE1T1Tx1`	If this is set it means that nothing is being processed by the SONIC E1/T1 framer. This is caused by faulty hardware.	status	3.7T
`kBooleanAttributeEquipment Loopback`	Enables/disables EQL. Useful for testing. Should normally be disabled.	config	3.7D, 3.7T, 3.8S, 4.3GE, 4.3S, 4.5G2, 4.5G4, 5.2SXA, 5.2X, 8.1SX, 8.1X, 8.2X,
`kBooleanAttributeFacility Loopback`	Enables/disables FCL. Useful for testing. Should normally be disabled.	config	3.7D, 3.7T, 3.8S, 4.3S, 5.2SXA, 5.2X, 8.1SX, 8.1X, 8.2X,
`kBooleanAttributeFault`	Indicates if there is a fault on the line. This is an "or" with `kBooleanAttributeTxFault` and `kBooleanAttributeRxFault`	status	5.2X, 8.2X,
`kBooleanAttributeFIFOLimitStatus`	Indicates that the jitter attenuator read/write FIFO pointers are within ± 3 bits.	status	3.7T,
`kBooleanAttributeFIFOError`	Tx FIFO error.	status	5.2SXA, 7.1S, 8.1SX, 8.1X,
`kBooleanAttributeFullDuplex`	Indicates if the link is full duplex	status	3.7G
`kBooleanAttributeHECCorrection`	Sets HEC correction on the port, only when the firmware supports ATM receive. Can be true (ON) or false (OFF).	config	3.7T,
`kBooleanAttributeHighBitError RateDetected.`	Indicates a high bit error rate has been detected. Check optical level (Ethernet only)	status	5.2X, 8.2X,
`kBooleanAttributeIDelay`	Indicates the IDELAY tap counter value.	status	
`kBooleanAttributeIDelay_Present`	Indicates the IDELAY mechanism to control the IDELAY tap value is present.	status	5.2SXA, 8.1SX, 8.1X,
`kBooleanAttributeJabber`	In an Ethernet network, jabber is traffic from a device that is always sending, bringing the network effectively to a halt. The attribute indicates whether jabber is being detected.	status	3.7G
`kBooleanAttributeLaser`	Enables/disables the transmit laser.	config	4.3S, 4.5G2, 4.5G4,
`kBooleanAttributeLineAlarm IndicationSignal`	Indicates if the Line Alarm Indication.	status	4.3S,
`kBooleanAttributeLineCode Violation`	Indicates the receiver channel is currently detecting a line code violation or an excessive number of zeros in the B8ZS or HDB3 mode.	status	3.7T,
`kBooleanAttributeLineRemote DefectIndicationSignal`	Indicates if the Line Remote Defect Indication Signal is set.	status	4.3S,
`kBooleanAttributeLineSide Equipment Loopback`	Enables/disables Line Side EQL. Useful for testing. Normally be disabled.	config	
`kBooleanAttributeLineSide FacilityLoopback`	Enables/disables Line Side FCL. Useful for testing. Should normally be disabled.	config	

Attribute	Description	Access	DAG Card
`kBooleanAttributeLink`	Indicates whether the link is Ok or not. In general terms, if there is synchronization then the link will be OK.	status	3.7G, 4.3GE, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.2X, 5.4SG48, 5.4GA, 5.4SGA48, 8.2X,
`kbooleanAttributeLinkDiscard`	When unset, packets/cells with checksum errors are passed through as if having no error. If set, the errored packets are dropped.	config	3.7D, 4.3S,
`kBooleanAttributeLocalFault`	Signal from peer is not being received correctly (Ethernet only).	status	
`kBooleanAttributeLock`	Indicates if the device is locked to the data stream.	status	5.2SXA, 5.2X, 7.1S, 8.1SX, 8.1X, 8.2X,
`kBooleanAttributeLossofCell Delineation`	Indicates if the demapper has lost cell delineation.	status	4.3S,
`kBooleanAttributeLossOfClock`	Indicates the framer is not receiving a valid clock from the optics.	status	
`kBooleanAttributeLossOfFrame`	Indicates if link is experiencing LOF.	status	3.7D, 3.8S, 4.3S, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.4SG48, 5.4GA, 5.4SGA48,
`kBooleanAttributeLossOfPointer`	Indicates if the link is experiencing Loss of Pointer ie. is unable lock to the SONET/SDH framer pointers (POS and WAN only).	status	3.8S, 4.3S,
`kBooleanAttributeLossOfSignal`	Indicates if link is experiencing LOS.	status	3.7D, 3.8S, 4.3S, 4.5G2, 4.5G4, 5.2SXA, 8.1SX, 8.1X,
`kBooleanAttributeMaster`	Indicates if the card is resolved to Master or Slave mode.	status	
`kBooleanAttributeNic`	Enables or disables Ethernet auto-negotiation mode. By default this is disabled. The disabled mode is intended for use with optical fibre splitters and in this mode auto-negotiation is not performed.	config	3.7G, 4.3GE, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.4SG48, 5.4GA, 5.4SGA48,
`kBooleanAttributeOutOfFrame`	Indicates the card is in an Out of Frame (OOF) condition.	status	3.7D, 3.8S, 4.3S,
`kBooleanAttributePayloadScramble`	Enables/disables payload scrambling for a concatenated POS demapper.	config	3.8S, 4.3S,
`kBooleanAttributePeerLink`	Indicates that the peer link is up and running.	status	4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.4SG48, 5.4GA, 5.4SGA48,
`kBooleanAttributePMaxCheck`	Enables/disables discard of packets larger than a pre-defined maximum size.	status	

Attribute	Description	Access	DAG Card
`kBooleanAttributePMinCheck`	Enables/disables discard of packets larger than a pre-defined minimum size	config	4.5G4, 5.0SG2, 5.0SG2A, 5.4SG48, 5.4GA, 5.4SGA48,
`kBooleanAttributePromisc`	Sets the card to Promiscuous mode	config	
`kBooleanAttributePromiscuousMode`	Ethernet promiscuous mode	config	4.3GE,
`kBooleanAttributeRDIError`	Indicates if the link is experiencing a remote data error.	status	3.8S,
`kBooleanAttributeReceive AlarmIndicaton`	Indicates a receive failure. If this attribute is set to "1" then either one or both of `kBooleanAttributeReceiveLockError` and `kBooleanAttributeReceivePowerAlarm` will also be set.	status	
`kBooleanAttributeReceive LockError`	Indicates a failure in clock recovery from the received signal.	status	3.7D, 5.2SXA, 8.1SX, 8.1X,
`kBooleanAttributeReceive LossOfSignal`	Indicates the receive input signal is lost.	status	3.7T,
`kBooleanAttributeReceive PowerAlarm`	Indicates insufficient optical input power (≤ 30dBm).	status	
`kBooleanAttributeReference OutOfLock`	Infdicates a ReferenceOut of Lock error condition.	status	4.3S,
`kBooleanAttributeREIError`	Indicates if the link is experiencing a remote error.	status	3.8S,
`kBooleanAttributeRemote DefectIndication`	Indicates if the Line Remote Defect Indication Signal is set.	status	3.7D
`kBooleanAttributeRemoteFault`	Indicates if there is a fault at the remote end of the link.	status	3.7G, 4.3GE, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.4SG48, 5.4GA, 5.4SGA48,
`kBooleanAttributeReset`	Holds/releases the framer in reset.	config	3.8S, 4.3S,
`kBooleanAttributeRocketIOPower`	Enables/disables the Rocket I/O.	config	4.5G2, 4.5G4,
`kBooleanAttributeRxFault`	Indicates a fault in the Rx data path.	status	5.2X, 8.2X,
`kBooleanAttributeRxLockError`	Indicates a Rx lock error is present on the PHY.	status	
`kBooleanAttributeRXMonitorMode`	Enable or disable RX Monitoring. This is used to enable the receive LIU monitor mode pre-amplifier. Enabling the pre-amplifier adds about 20 dB of linear amplification for use in monitor applications where the signal has been reduced 20 dB using resistive attenuator circuits.	config	3.7D
`kBooleanAttributeRxPkts`	Enables or disables receive packets.	config	3.7T, 4.3GE,
`kBooleanAttributeScramble`	Enables/disables SONET frame scrambling.	config	3.8S, 4.3S,
`kBooleanAttributeSFPDetect`	Indicates if the SFP is present.	status	4.5G2, 4.5G4,
`kBooleanAttributeSfpPower`	Enables/disables optics transmit power.	config	4.5G2,
`kBooleanAttributeSync`	Indicates the synchronization status.	status	4.3GE,
`kBooleanAttributeTxCrc`	Enables/disables CRC appending onto transmitted frames.	status	4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.4SG48, 5.4GA, 5.4SGA48,
`kBooleanAttributeTxFault`	Indicates a fault in the TX data path	status	5.2X, 8.2X,
`kBooleanAttributeTxFIFOError`	Indicates a TX FIFO error.	status	
`kBooleanAttributeTxLockError`	Indicates a TX lock error is present on the PHY.	status	
`kBooleanAttributeTxPkts`	Enables or disables transmit packets.	config	3.7T, 4.3GE,
`kNullAttributeDefaultDS3ATM`	Configure the card into DS3 framed full line rate ATM mode.	config	3.7D

Attribute	Description	Access	DAG Card
kNullAttributeDefaultDS3HDLC	Configure the card into DS3 framed full line rate HDLC mode.	config	3.7D
kNullAttributeDefaultE3ATM	Configure the card into E3 framed full line rate ATM mode.	config	3.7D
kNullAttributeDefaultE3HDLC	Configure the card into E3 framed full line rate HDLC mode.	config	3.7D
kNullAttributeDefaultKentrox	Configure the card into Kentrox mode.	config	3.7D
kStructAttributeErfMux	Sets packet steering between Host, Line and Xscale. For more information see `erf_mux_37t_t`.	config	3.7T, 7.1S
kUint32AttributeAborts	Number of POS frames aborted since last reading.	status	4.3S,
kUint32AttributeB1ErrorCount	Bit Interleaved Parity 1. SONET/SDH section parity error count (POS only).	status	
kUint32AttributeB2ErrorCount	Bit Interleaved Parity 2. SONET/SDH section parity error count (POS only).	status	
kUint32AttributeB3ErrorCount	Bit Interleaved Parity 2. SONET/SDH section parity error count (POS only).	status	
kUint32AttributeBadSymbol	Count of the number times a valid length frame was received at the port and during which time there was at east one of an event that causes the PHY to indicate a “Data Reception Error” or in valid “Data Symbol Error”	status	
kUint32AttributeBERCounter	Count of the number of High Bit Errors seen.	status	5.2X, 8.2X,
kUint32AttributeC2PathLabel	Reads the SONET/SDH C2 path label byte (Path Signal Label). Typical settings for POS are 0x16, for Cisco HDLC is 0xCF. On cards that support virtual containers the path label will be read from the container specified by `kUint32AttributeVCIndex` attribute.	status	3.8S, 4.3S,
kUint32AttributeCableLoss	A number representing the cable attenuation indication within ±1dB.	status	3.7T,
kUint32AttributeCrcSelect	Sets the CRC to use. For valid CRC types.	config	3.7D, 3.8S, 4.3S, 4.5G4, 5.0SG2, 5.0SG2A, 5.4SG48, 5.4GA, 5.4SGA48,
kUint32AttributeDropCount	Count of the number of packets dropped on a port since last card reset.	status	3.7D, 3.7G, 3.8S, 4.3GE, 4.3S, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.2X, 5.4SG48, 5.4GA, 5.4SGA48, 8.2X,
kUint32AttributeErrorBlock Counter	Count of the number of error blocks that have occurred since this attribute was last read using an MDIO. This has a maximum value of 255 and does not rollover to) when further error blocks are detected. It is cleared to 0 when read.	status	5.2X, 8.2X,
kUint32AttributeErrorCounter	Use this attribute to count the errors.	status	3.7G
kUint32AttributeEthernetMode	Configures the Ethernet mode on the card. Must be `kEthernetMode10GBase_SR` or `kEthernetMode10GBase_LR` or `kEthernetMode10GBase_ER`. Sets the port to LAN or WAN mode.	status`	5.2X, 8.2X,
kUint32AttributeForceLineRate	Forces the card to operate at the given line rate.	config	3.7G
kUint32AttributeFramingMode	Indicates the type of framing to be used. These modes are listed in the enumeration framing_mode_t.	config	3.7D

Attribute	Description	Access	DAG Card
`kUint32AttributeHECCount`	Count of the number of cells with HEC error since this attribute was last read.	status	4.3S,
`kUint32AttributeIDELAY`	IDELAY Tap Counter Value	config	5.2SXA, 8.1SX, 8.1X,
`kUint32AttributeLineRate`	Sets the card line rate. The card can operate at OC-3 or OC-12.	config	3.7G, 3.8S, 4.3GE, 4.3S, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.4SG48, 5.4GA, 5.4SGA48, 7.1S,
`kUint32AttributeLineType`	Set the line type. For valid values see `line_type_t`.	config	3.7G, 3.7T,
`kUint32AttributeMacAddress`	Retrieves the MAC Address for the Ethernet port.	status	
`kUint32AttributeMasterSlave`	Sets the SONET clock master/slave status. For valid values see `master_slave_t`.	config	3.7G, 4.3S, 5.2SXA, 7.1S, 8.1SX, 8.1X,
`kUint32AttributeMaxPktLen`	Sets the maximum expected packet length and maximum size for `kBooleanAttributePMax Check`.	config	4.3GE, 4.3S,
`kUint32AttributeMaxPktLenError`	Count of the number of packets rejected because they were too large since the last time this attribute was read.	status	4.3S,
`kUint32AttributeMem`	The memory allocated to a transmit or receive stream. Can be used to allocate different amounts of memory from the buffer to a stream. The size of the buffer can be read using the attribute `kUint32AttributeBuffer Size`. The value is returned in Mebi-bytes.	config	3.7D, 3.7G, 3.7T, 3.8S, 4.3GE, 4.3S, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.2SXA, 5.2X, 5.4SG-48, 5.4SGA-48, 7.1S, 8.1SX, 8.1X, 8.2X, 8.4I
`kUint32AttributeMemBytes`	Same as above except the unit of measurement is bytes.	config	
`kUint32AttributeMinPktLen`	Sets the minimum expected packet length and minimum size for `kBooleanAttributePMinCheck`.	config	4.3S,
`kUint32AttributeMinPktLenError`	Count of the number of packets rejected because they were too small since the last time this attribute was read	status	4.3S,
`kUint32AttributeMode`	For more information see Mode Table.	config	3.7T,
`kUint32AttributeNetworkMode`	Sets the port to POS or ATM mode. See `network_mode_t`.		3.7D, 3.8S, 4.3S,
`kUint32AttributePathBIPError`	Count of the number of Path Bit Interleaved Parity Errors seen.	status	4.3S,
`kUint32AttributePathREIError`	Count of the number of Path Remote Error indications seen.	status	4.3S,
`kUint32AttributePayloadMapping`	Determines the type of payload mapping see section `payload_mapping_t.`	config	
`kUint32AttributeRxFDrop`	Count of the number of receives frames dropped since the last time this attribute was read.	status	4.3S,
`kUint32AttributeRxFrames`	Count of the number of valid frames received since the last time this attribute was read.	status	3.7D, 4.3S,
`kUint32AttributeRxParityError`	Indicates there is a receive parity error count between the framer and receive FPGA (PoS) only)	status	
`kUint32AttributeSteer`	Set the ERF record steering mode. See `steer_t`.	config	
`kUint32AttributeTermination`	Sets the termination strength. For valid values see `termination_t`.	config	3.7T,

Attribute	Description	Access	DAG Card
`kUint32AttributeTxFDrop`	Count of the number of frames dropped during transmission since the last time this attribute was read.	status	4.3S,
`kUint32AttributeTxFrames`	Count of the number of valid frames transmitted since the last time this attribute was read.	status	3.7D, 4.3S,
`kUint32AttributeZeroCodeSuppress`	For valid values see zero_code_suppress_t.	config	3.7T,
`kUint64AttributeBadPackets`	Count of the number of errored packets/frames received since this attribute was last read.	status	
`kUint64AttributeBadSymbol`	Count of the number times a valid length frame was received at the port and during which time there was at east one of an event that causes the PHY to indicate a "Data Reception Error" or in valid "Data Symbol Error"	status	4.3GE,
`kUint64AttributeCrcFail`	Count of the number of frames received that do not pass the Frame Checksum (FCS) check.	status	4.3GE,
`kUint64AttributeFCSErrors`	Count of the number of PoS/Ethernet FCS (CRC32) errors	status	
`kUint64AttributeFIFOOverrunCount`	Framer receives FIFO errors since the last time this attribute was read.	status	
`kUint64AttributeGoodPackets`	Count of the number of packets/frames received since this attribute was last read.	status	
`kUint64AttributeInternalMACError`	Count of the number of frames that could not be sent because of various errors.	status	4.3GE,
`kUint64AttributeRxBytes`	Count of the number of bytes successfully received.	status	3.7D, 4.3GE, 4.3S,
`kUint64AttributeRxBytesBad`	Count of the number of errored bytes received.		
`kUint64AttributeRxFrames`	Count of the number of valid frames received.	status	4.3GE,
`kUint64AttributeTransmit SystemError`	Count of the number of frames that could not be sent correctly because of various errors. Frames that have already been counted by the attribute `kUint64AttributeInternalMACError` are not included in this count.	status	4.3GE,
`kUint64AttributeTxBytes`	Count of the number of bytes successfully transmitted.	status	3.7D, 4.3GE, 4.3S,
`kUint64AttributeTxFrames`	Count of the number of frames successfully transmitted.	status	4.3GE,

kComponentSC256

Represent the SC256 component

4.3GE

Attribute	Description	Access
`kNullAttributeSC256Init`	Initializes the SC256 Coprocessor.	config
`kStructAttributeSC256144BitSearch`	Use this attribute to perform 144-bit searches.	config
`kStructAttributeSC25672BitData`	Use this attribute to read/write data to the TCAM.	config
`kStructAttributeSC25672BitMask`	Use this attribute to read/write mask values to the TCAM.	config
`kStructAttributeSC25672BitSearch`	Use this attribute to perform 72-bit searches.	config
`kUint32AttributeSC256DataAddress`	Use this attribute to set the address of the data space on the CAM to read or write.	config
`kUint32AttributeSC256MaskAddress`	Use this attribute to set the address of the mask space on the CAM to read or write.	config
`kUint32AttributeSC256SearchLength`	Use this attribute to set the search length. /sa SC256SearchLength.	config

kComponentSonetPP

SONET Packet processor component on the new cards.

5.0SG2, 5.0SG2A, 5.2SXA, 5.4S-12, 5.4SG-48, 5.4SA-12, 5.4SGA-48

Attribute	Description	Access
`kBooleanAttributeAlarmIndicationSignal`	Indicate the card is experiencing an alarm signal.	status
`kBooleanAttributeB1Error`	SONET B1 error has occurred.	status
`kBooleanAttributeB2Error`	SONET B2 error has occurred.	status
`kBooleanAttributeB3Error`	SONET B3 error has occurred.	status
`kBooleanAttributeCounterLatch`	This attribute must be set before reading statistics. It latches the statistics counters to allow values to be read.	config
`kBooleanAttributeRxFIFOEmpty`	Indicates the receive FIFO is empty.	status
`kBooleanAttributeTXFIFOFull`	Indicates the transmit FIFO is full.	status
`kBooleanAttributeTXFIFOOverflow`	Indicates the transmit FIFO is overflowing.	status
`kBooleanAttributeLossOfFrame`	Indicates the card is experiencing loss of frame.	status
`kBooleanAttributeLossOfPointer`	Indicates the card is experiencing loss of pointer.	status
`kBooleanAttributeLossOfSignal`	Indicates the card is experiencing loss of signal.	status
`kBooleanAttributeSonetMode`	Selects between SONET (1) or SDH (0).	config
`kBooleanAttributeOutOfFrame`	Indicates the card is experiencing an out of frame error.	status
`kBooleanAttributePayloadScramble`	Enables/disables POS scrambling.	config
`kBooleanAttributeRDIError`	Indicates a remote data error.	
`kBooleanAttributeRefreshCache`	Latch the extended statistics.	config
`kBooleanAttributeREIError`	Indicates a remote error.	status
`kBooleanAttributeScramble`	Enables /disables SONET frame scrambling.	config
`kUint32AttributeB1ErrorCount`	The numbers of B1 errors which have occurred since last read.	status
`kUint32AttributeB2ErrorCount`	The numbers of B2 errors which have occurred since last read.	status
`kUint32AttributeB3ErrorCount`	The numbers of B3 errors which have occurred since last read.	status
`kUint32AttributeC2PathLabel`	The C2 path label.	status
`kUint32AttributeCrcSelect`	Selects the CRC size.	config
`kUint32AttributeCRCError`	Indicates a CRC error.	status
`kUint32AttributeJ0Path Label`	The path trace value of the indicated virtual container and byte of the data pointer.	status
`kUint32AttributeJ1Path Label`	The path trace value of the indicated virtual container	status

	and byte of the data pointer.	
`kUint32AttributeLineRate`	The rate at which the line is currently operating.	config
`kUint32AttributeNetworkMode`	Selects POS or ATM mode.	config
`kUint32AttributeREIErrorCount`	The number of time a remote error has occurred.	status

kComponentSonic

Controls attributes of the SONET/SDH deframer.

7.1S

Attribute	Description	Access
`kBooleanAttributeB1Error`	SONET B1 Error indication.	status
`kBooleanAttributeB2Error`	SONET B2 Error indication.	status
`kBooleanAttributeB3Error`	SONET B2 Error indication.	status
`kBooleanAttributeCounterLatch`	This attribute must be set before reading statistics. It latches the statistics and counters to allow values to be read.	config
`kBooleanAttributeLossOfFrame`	Indicates that the card is experiencing Loss Of Frame.	status
`kBooleanAttributeLossOfSignal`	Indicates that the card is experiencing Loss Of Signal.	status
`kBooleanAttributeOutOfFrame`	Indicates if link is experiencing OOF.	status
`kBooleanAttributeRDIError`	Indicates that the SONET Remote Data Indication is set.	status
`kBooleanAttributeRefreshCache`	Refresh the cache that stores the SDH status.	config
`kBooleanAttributeREIError`	Indicates that the SONET Remote Error Indication is set.	status
`kBooleanAttributeScramble`	Use to enable or disable SONET frame scrambling.	config
`kUint32AttributeB1ErrorCount`	The number of B1 errors since last read.	status
`kUint32AttributeB2ErrorCount`	The number of B2 errors since last read.	status
`kUint32AttributeB3ErrorCount`	The number of B3 errors since last read.	status
`kUint32AttributeC2PathLabel`	The C2 path label of the virtual container indicated by the `Uint32AttributeVCIndex.`	status
`kUint32AttributeConnectionNumber`	The connection number of the current configuration.	config
`kUint32AttributeConnectionVCLabel`	Retrieve the VC Label for the connection specified.	status
`kUint32AttributeConnectionVCPointer`	Retrieve the VC pointer for the connection specified	status
`kUint32AttributeDataPointer`	Which data byte to read for `kUint32AttributeJ0PathLabel` and `kUint32AttributeJ1PathLabel.`	config
`kUint32AttributeDeframerRevisionID`	Retrieve the revision id of the deframer	status
`kUint32AttributeERDIError`	e-rdi_error error has occurred.	status
`kUint32AttributeJ0PathLabel`	The section trace value of the indicated virtual container and byte of the data pointer.	status
`kUint32AttributeJ1PathLabel`	The path trace value of the indicated virtual container and byte of the data pointer.	status
`kUint32AttributeLineRate`	The rate at which the line is currently operating.	config
`kUint32AttributePayLoadMapping`	Determines the type of payload mapping.	config
`kUint32AttributePointerState`	The pointer state of the various virtual containers..	status
`kUint32AttributeREIErrorCount`	Number of remote error indications seen.	status
`kUint32AttributeSSM`	The received synchronization status message.	status
`kUint32AttributeTributaryUnit`	Defines the type of payload to extract. See section `tributary_unit_t.`	config
`kUint32AttributeTXC2PathLabel0`	Attribute to set and get transmit C2 byte at index 0.	config
`kUint32AttributeTXC2PathLabel1`	Attribute to set and get transmit C2 byte at index 1.	config
`kUint32AttributeTXC2PathLabel2`	Attribute to set and get transmit C2 byte at index 2.	config
`kUint32AttributeTXC2PathLabel3`	Attribute to set and get transmit C2 byte at index 3.	config
`kUint32AttributeTXC2PathLabel4`	Attribute to set and get transmit C2 byte at index 4.	config
`kUint32AttributeTXC2PathLabel5`	Attribute to set and get transmit C2 byte at index 5.	config
`kUint32AttributeTXC2PathLabel6`	Attribute to set and get transmit C2 byte at index 6.	config

`kUint32AttributeTXC2PathLabel7`	Attribute to set and get transmit C2 byte at index 7.	config
`kUint32AttributeTXC2PathLabel8`	Attribute to set and get transmit C2 byte at index 8.	config
`kUint32AttributeTXC2PathLabel9`	Attribute to set and get transmit C2 byte at index 9.	config
`kUint32AttributeTXC2PathLabel10`	Attribute to set and get transmit C2 byte at index 10.	config
`kUint32AttributeTXC2PathLabel11`	Attribute to set and get transmit C2 byte at index 11.	config
`kUint32AttributeTXV5SignalLabel`	TX V5 Signal Label	config
`kUint32AttributeVCIndex`	Retrieve or specify the index of the virtual container to use. Any index written should be less than the result of reading `kUint32AttributeVCMaxIndex.`	config
`kUint32AttributeVCMaxIndex`	The maximum number of active virtual containers in the SONET frame. This number depends on the hardware, loaded firmware and virtual container size.	status
`kUint32AttributeVCSize`	Use this attribute to set or get the size of the Virtual Containers.	config

kComponentSteering

The steering component. This allows one to choose a algorithm to steer the received packets. The steering algorithm allows the packets to be directed to different memory holes depending on for example a crc hash function.

4.5G2, 4.5G4, 5.0SG2, 5.2X, 5.4S-12, 5.4SG-48, 5.4GA, 5.4SA-12, 8.2X

Attribute	Description	Access
`kUint32AttributeSteer`	The algorithm to use to steer the incoming packet.	config

kComponentStream

The stream component models a receive stream or transmit stream. The number of streams depends on the loaded firmware image. This component can be used to allocate memory to the stream it represents.

3.7D, 3.7G, 3.7T, 3.8S, 4.3GE, 4.3S, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.2SXA, 5.2X, 5.4S-12, 5.4SG-48, 5.4GA, 5.4SA-12, 5.4SGA-48, 7.1S, 8.1SX, 8.1X, 8.2X, 8.4I

Attribute	Description	Access
`kUint32AttributeLimitPointer`	The point where the next read will take place. This pointer is updated by the DAG API.	status
`kUint32AttributeMem`	The amount of memory (in MiB) allocated to a transmit or receive stream. Writing to this attribute allocate specific amount of memory from the buffer to an individual stream. The size of the buffer can be read using the attribute `kUint32AttributeBufferSize.`	config
`kUint32AttributeMemBytes`	Same as above except the unit of measurement is bytes.	config
`kUint32AttributeRecordPointer`	The point where the next record will be written. This is updated by firmware each time it writes to the memory hole.	status

kComponentStreamFeatures

Component for new per stream features like snap length per stream

5.0SG2A,

Attribute	Description	Access
`kBooleanAttributeSLEN_Present`	Per stream snap length is present or not.	status
`kUint32AttributeMaxSnapLen`	Maximum allowed Snap length for stream.	status
`kUint32AttributeNumberOfRegisters`	Number of registers per stream.	status
`kUint32AttributeNumberOfStreams`	Number of streams supported.	status
`kUint32AttributeSLen`	Snap length for stream.	config

kComponentTerf

Represents the `terf` register on cards that have the appropriate firmware loaded

3.7G, 3.8S, 4.5G2, 4.5G4, 5.0SG2, 5.0SG2A, 5.2X, 5.4S-12, 5.4SG-48, 5.4SGA-48, 7.1S, 8.1SX, 8.2X,

Attribute	Description	Access
`kUint32AttributeTerfStripCrc`	Sets the number of bytes to strip from the end of the ERF record when transmitting. Used to prevent a trailing CRC (e.g. on an ERF header that has been captured and is now being transmitted) being sent as part of a packet.	config

4.5G2, 5.0SG2, 5.0SG2A, 5.2SXA, 5.2X, 5.4S-12, 5.4SG-48, 5.4SGA-48, 8.1SX, 8.2X,

Attribute	Description	Access
`kBooleanAttributeRXErrorA`	Mask for the RX-Error bit in ERF header.	config

4.5G2, 5.0SG2, 5.0SG2A, 5.4S-12, 5.4SG-48, 5.4SGA-48,

Attribute	Description	Access
`kBooleanAttributeRXErrorB`	Mask for the RX-Error bit in ERF header for the 2nd output port if available.	Config

4.5G4,

Attribute	Description	Access
`kBooleanAttributeRXErrorC`	Mask for the RX-Error bit in ERF header.	config
`kBooleanAttributeRXErrorD`	Mask for the RX-Error bit in ERF header for the 2nd output port if available.	Config

4.5G4, 5.2SXA, 5.4SG-48, 5.4SGA-48,

Attribute	Description	Access
`kBooleanAttributeShiftDirection`	Determines the shift direction, left or right.	config
`kUint32AttributeScaleRange`	Number of logical shifts performed between 2 packet's timestamps.	config
`kUint32AttributeTimeMode`	Determine the timed release mode.	config
`kUint64AttributeAbsModeOffset`	Absolute mode offset value.	config
`kUint64AttributeConfLimit`	The conf limit value	config

kComponentXGMII

The XGMII component.

5.2SXA, 5.2X, 8.1SX, 8.1X, 8.2X,

Attribute	Description	Access
`kBooleanAttributeCrcStrip`	Enable or disable CRC stripping from received frames.	config
`kBooleanAttributeTxCrc`	Enable or disable CRC appending onto transmitted frames.	config
`kStringAttributeEthernetMACAddress`	MAC address for Ethernet port.	status
`kUint32AttributeNetworkMode`	POS	config

Chapter 4: DAG 3.7T Card

Structures

erf_mux_37t_t

Description

To direct packet steering for the 3.7T card.

```
typedef struct
{
      uint32_t mHost;
      uint32_t mLine;
      uint32_t mXscale;
} erf_mux_37t_t;
```

The packets can be steered in the directions defined in erf_mux_steering_37t_t.

```
typedef enum
{
    kErfToHost = 0x0,
    kErfToLine = 0x01,
    kErfToXscale = 0x02

} erf_mux_steering_37t_t;
```

Include

```
dag_attribute_codes.h
```

connection_description_37t_t

Description

To describe the connection to be added to the 3.7T card.

```
typedef struct
{
    connection_type37t_t mConnectionType;
    payload_type_t mPayloadType;
    direction_t mDirection;
    uint32_t mline;
    uint32_t mTimeslot;
    uint32_t mMask;
    uint32_t mConnectionNumber;
} connection_description_37t_t;
```

Include

`dag_attribute_codes.h`

Member	Type	Description
`mConnectionType`	`connection_type37t_t`	Describes the type of connection to be added (i.e. single channel, hyper channel, sub channel or raw types)
`mPayloadType`	`payload_type_t`	Sets the payload type, when multiple types are available in the firmware (ATM and HDLC firmware types).
`mDirection`	`direction_t`	Sets the direction of the channel (either transmit or receive). The transmit setting requires transmit capable firmware to be loaded into the card.
`mline`	32 bit unsigned integer	The physical line (port) the connection is to be added to. (0-15)
`mTimeslot`	32 bit unsigned integer	Timeslot to create channel on when creating a single timeslot connection only, such as a single channel or sub channel connection. This should be zero when adding a hyper channel. Valid timeslot values are between 1-31 for channelized E1 or 1-24 for T1.
`mMask`	32 bit unsigned integer	Mask to define the timeslot mask for hyper channel or the bit mask for sub channel connections. For sub channel connections the mask should be no greater than 255 (8 bit value). For hyper channel connections this mask represents the timeslots to be used, where each bit represents the corresponding timeslot, i.e. bit zero is timeslot zero, bit one is timeslot one, the first valid timeslot for channelized operation. Valid timeslot values are between 1-31 for channelized E1 or 1-24 for T1 and 0-31 for unchannelized E1.
`mConnectionNumber`	32 bit unsigned integer	The unique number of the connection as set by the system. This value can be subsequently used in deletion operations.

Notes on the Connection Components

Layout:

Mapper/Demapper Component

Revision attribute (when applicable)

Other possible global mapper and demapper attributes

Add connection (receives structure with channel details) attribute. Returns the connection number in a member of the structure received.

Get last connection number attribute

Delete connection (receives connection number) attribute

Delete all connections attribute

Get connection count attribute

Connection 0 Subcomponent (same numbering as connection number)

Get active/inactive attribute (read only)

Get Connection Information attribute (only available when connection active. Information available will depend on the card in use)

Connection 1 Subcomponent

...

Connection n Subcomponent

The structure to be used when creating the connection and the available attributes of the connection subcomponent depend on the card in use. The structure used for adding a connection for each card will now contain a member which will hold the connection number. The connection number will be allocated by the system when a connection is added successfully. This connection number given by the system will also correspond to the connection subcomponent number.

Information available about an active connection is dependent on the card in use and is detailed in the card specific section below.

Mode Table

The line characteristics associated with each supported mode are shown below.

Mode	Type	Tx LBO	Cable	Coding
0	T1 Long Haul/36dB	0dB	100Ω/ TP	B8ZS
1	T1 Long Haul/36dB	-7.5dB	100Ω/ TP	B8ZS
2	T1 Long Haul/36dB	-15dB	100Ω/ TP	B8ZS
3	T1 Long Haul/36dB	-22.5dB	100Ω/ TP	B8ZS
4	T1 Long Haul/45dB	0dB	100Ω/ TP	B8ZS
5	T1 Long Haul/45dB	-7.5dB	100Ω/ TP	B8ZS
6	T1 Long Haul/45dB	-15dB	100Ω/ TP	B8ZS
7	T1 Long Haul/45dB	-22.5dB	100Ω/ TP	B8ZS
8	T1 Short Haul/15dB	0-133 ft./ 0.6dB	100Ω/ TP	B8ZS
9	T1 Short Haul/15dB	133-266 ft./ 1.2dB	100Ω/ TP	B8ZS
10	T1 Short Haul/15dB	266-399 ft./ 1.8dB	100Ω/ TP	B8ZS
11	T1 Short Haul/15dB	399-533 ft./ 2.4dB	100Ω/ TP	B8ZS
12	T1 Short Haul/15dB	533-655 ft./ 3.0dB	100Ω/ TP	B8ZS
13	T1 Short Haul/15dB	Arbitrary Pulse	100Ω/ TP	B8ZS
14	T1 Gain Mode/29dB	0-133 ft./ 0.6dB	100Ω/ TP	B8ZS
15	T1 Gain Mode/29dB	133-266 ft./ 1.2dB	100Ω/ TP	B8ZS
16	T1 Gain Mode/29dB	266-399 ft./ 1.8dB	100Ω/ TP	B8ZS
17	T1 Gain Mode/29dB	399-533 ft./ 2.4dB	100Ω/ TP	B8ZS
18	T1 Gain Mode/29dB	533-655 ft./ 3.0dB	100Ω/ TP	B8ZS
19	T1 Gain Mode/29dB	Arbitrary Pulse	100Ω/ TP	B8ZS
20	T1 Gain Mode/29dB	0dB	100Ω/ TP	B8ZS
21	T1 Gain Mode/29dB	-7.5dB	100Ω/ TP	B8ZS
22	T1 Gain Mode/29dB	-15dB	100Ω/ TP	B8ZS
23	T1 Gain Mode/29dB	-22.5dB	100Ω/ TP	B8ZS
24	E1 Long Haul/36dB	ITU G.703/Arbitrary	75Ω/ coax	HDB3
25	E1 Long Haul/36dB	ITU G.703/Arbitrary	120Ω/ TP	HDB3
26	E1 Long Haul/43dB	ITU G.703/Arbitrary	75Ω/ coax	HDB3
27	E1 Long Haul/43dB	ITU G.703/Arbitrary	120Ω/ TP	HDB3
28	E1 Short Haul	ITU G.703/Arbitrary	75Ω/ coax	HDB3
29	E1 Short Haul	ITU G.703/Arbitrary	120Ω/ TP	HDB3
30	E1 Gain Mode	ITU G.703/Arbitrary	75Ω/ coax	HDB3

Enumerations

connection_type37t_t

Description

Defines the type of connection to add to the 3.7T card.

```
typedef enum
{
    kConnectionTypeNULL     = 0x0,
    kConnectionTypeChan     = 0x1,
    kConnectionTypeHyper    = 0x2,
    kConnectionTypeSub      = 0x3,
    kConnectionTypeRaw      = 0x4,
    kConnectionTypeChanRaw  = 0x5,
    kConnectionTypeHyperRaw = 0x6,
    kConnectionTypeSubRaw   = 0x7
} connection_type37t_t;
```

Include

`dag_attribute_codes.h`

payload_type_t

Description

Defines the payload type of connection to add to the 3.7T card. This will define the module to be used when mixed mode firmware is loaded.

```
typedef enum
{
    kPayloadTypeNotConfigured = 0x0,
    kPayloadTypeATM = 0x01,
    kPayloadTypeHDLC = 0x02,
    kPayloadTypeRAW = 0x05
} payload_type_t;
```

Include

`dag_attribute_codes.h`

direction_t

Description

Defines the direction of connection to add to the 3.7T card. This will define the module to be used when transmit firmware is loaded.

```
typedef enum
{
    kDirectionUndefined = 0x0,
    kDirectionReceive = 0x01,
    kDirectionTransmit = 0x02
} direction_t;
```

Include

`dag_attribute_codes.h`

Chapter 5: DAG7.1S Card

Structures

connection_description_t

Description

To describe the connection to be added to the 7.1S card.

```
typedef struct
{
    uint32_t mTUG3_ID;
    uint32_t mVC_ID;
    uint32_t mTUG2_ID;
    uint32_t mTU_ID;
    uint32_t mPortNumber;
    connection_type_t mConnectionType;
    payload_type_t mPayloadType;
    uint8_t mScramble;
    uint8_t mHECCorrection;
    uint8_t mIdleCellMode;
    uint32_t mTimeslotMask;
    uint32_t mConnectionNumber;
} connection_description_t;
```

Include

`dag_attribute_codes.h`

Chapter 6: Card Configuration Functions

Overview

All of the functions defined in this chapter relate to functions which directly configure the DAG card.. They are listed in alphabetical order.

Other functions relating to components, modifiers, accessors and firmware are described later in this programming guide.

The following designators are used in card configuration functions:

Designator	Description
`card`	Refers to a DAG card.
`uuid`	An attribute identifier.
`component`	Refers to a component.
`device name`	The name of the card. In Linux this should look like `/dev/dag0` and in Windows® like `dag0`.
`string`	The value for the string in attribute form.

Functions

dag_config_default

Description

Executes a card's default configuration routine.

```
dag_err_t
dag_config_default
(
dag_card_ref_t card
);
```

Return Value

Returns `kDagErrInvalidCardRef` if the card reference is invalid

dag_config_dispose

Description

Cleans up when finished with a card reference.

```
void
dag_config_dispose
(
   dag_card_ref_t card
);
```

Return Value

None

dag_config_get_attribute_code

Description

Retrieves the attribute code of a given attribute.

```
dag_attribute_code_t
dag_config_get_attribute_code
(
   attr_uuid_t uuid
);
```

Return Value

The attribute code.

dag_config_get_attribute_config_status

Description

```
dag_attr_config_status_t
dag_config_get_attribute_config_status
(
   attr_uuid_t uuid
);
```

Return Value

See `dag_attrubute_config_status_t` for more information

dag_config_get_attribute_description

Description

```
const char*
dag_config_get_attribute_description
(
   attr_uuid_t uuid
);
```

Return Value

The description for the given attribute.

dag_config_get_attribute_name

Description

Retrieves a human-readable name for an attribute.

```
const char*
dag_config_get_attribute_name
(
   attr_uuid_t uuid
);
```

Return Value

The name of the given attribute.

dag_config_get _attribute_to_string

Description

Retrieves the value of an attribute as a string.

```
const char*
dag_config_get_attribute_to_string
(
   dag_card_ref_t card,
   attr_uuid_t uuid
);
```

Return Value

A string representing the value of the given attribute.

dag_config_get_attribute_valuetype

Description

Retrieves the type of an attribute's value.

```
const char*
dag_config_get_attribute_valuetype
(
   dag_card_ref_t card,
   attr_uuid_t uuid
);
```

Return Value.

The type of the given attribute's value.

dag_config_get_card_type

Description

Returns the type of card.

```
dag_card_t
dag_config_get_card_type
(
dag_card_ref_t card
);
```

Note: The type code is shown `dag-card_t`.

Return Value

The type of card.

dag_config_get_component_count

Description

Retrieves the number of components in the card.

```
int
dag_config_get_component_count
(
   dag_card_ref_t card
);
```

Return Value

The number of components.

dag_config_get_component_description

Description

Retrieves a humanly readable description for a component.

```
const char*
dag_config_get_component_description
(
   dag_component_t component
);
```

Return Value

The description for the given component

dag_config_get_component_name

Description

Retrieves a humanly readable name for a com ponent.

```
const char*
dag_config_get_component_name
(
   dag_component_t component
);
```

Return Value

The name of the given component

dag_config_get_root_component

Description

Retrieves the root component, from which all subcomponents descend, for a given card.

```
dag_component_t
dag_config_get_root_component
(
dag_card_ref_t card
);
```

Return Value

A reference to a root component.

dag_config_init

Description

Initializes the DAG card and retrieves a reference to the card for use with other functions in the API.

```
dag_card_ref_t
dag_config_init
(
const char* device_name
);
```

Once finished with the card, use dag_config_dispose to deallocate memory used internally by the API.

Return Value

A reference to a DAG card. `NULL` is returned on failure.

dag_config_reset

Description

Executes a card's reset configuration routine.

```
dag_err_t
dag_config_reset
(
dag_card_ref_t card
);
```

Return Value

`kDagErrInvalidCardRef` if the card reference is invalid.

dag_config_set_attribute_from_string

Description

Sets the value for an attribute from a string.

```
const char*
dag_config_get_attribute_from_string
(
   dag_card_ref_t card,
   attr_uuid_t uuid
   const char* string
);
```

Return Value

The value for the attribute

Chapter 7: Component Functions

Overview

All of the functions defined in this chapter relate to functions which configure or retrieve components on the DAG card. They are listed in alphabetical order.

Other functions relating to directly configuring the DAG card, modifiers, accessors and firmware are described in the previous and subsequent chapters of this programming guide.

The following designators are used in component functions:

Designator	Description
`attribute`	The code of the attribute to retrieve.
`component`	Refers to a component.
`component code`	See the card specific chapters earlier in this programming guide for a list of valid component codes.
`index`	The index of the attribute to return.
`name`	The name of the sub-component to return.
`code`	The desired subcomponent to count.

Functions

dag_component_get_attribute_count

Description

Retrieves the number of attributes in a component.

```
int
dag_component_get_attribute_count
(
   dag_component_t
);
```

Return Value

The number of attributes in that component.

dag_component_get_config_attribute_count

Description

Retrieves the number of config attributes in a given component

```
int
dag_component_get_config_attribute_count
(
   dag_component_t component
);
```

Return Value

The count of the number of config attributes.

dag_component_get_config_attribute_uuid

Description

Retrieves an attribute from a DAG component.

```
attr_uuid_t
dag_component_get_config_attribute_uuid
(
   dag_component_t component,
   dag_attribute_code_t attribute_code
);
```

Return Value

An identifier for the attribute if found. If the requested attribute cannot be found `kNullAttributeUuid` is returned.

dag_component_get_indexed_attribute_uuid

Description

Retrieves the DAG component attribute "`i`" at the given index.

```
attr_uuid_t
dag_component_get_indexed_attribute_uuid
(
   dag_component_t component,
   int index
);
```

Return Value

The attribute at the given index.

dag_component_get_indexed_config_attribute_uuid

Description

Retrieves a configuration attribute from a component by index.

```
attr_uuid_t
dag_component_get_indexed_config_attribute_uuid
(
dag_component_t component,
int index
);
```

Return Value

The configuration attribute at the given index.

dag_component_get_indexed_status_attribute_uuid

Description

Retrieves the status attribute at a given index.

```
attr_uuid_t
dag_component_get_indexed_status_attribute_uuid
(
   dag_component_t component,
   int index
);
```

Return Value

The status attribute at the given index.

dag_component_get_indexed_subcomponent

Descritpion

Retrieves a subcomponent at a given index.

```
attr_uuid_t
dag_component_get_indexed_subcomponent
(
dag_component_t component,
int index
);
```

Return Value

The subcomponent at the given index.

dag_component_get_named_subcomponent

Description

Retrieves the component using the internal name of the component

```
dag_component_t
dag_component_get_named_subcomponent
(
dag_component_t component,
const char* name
);
```

Return Value

The component or `NULL` if not found.

dag_component_get_status_attribute_count

Description

Retrieves the number of status attributes in a component.

```
int
dag_component_get_status_attribute_count
(
   dag_component_t component
);
```

Return Value

The number of status attributes

dag_component_get_subcomponent

Description

Retrieves a specific subcomponent of a given component.

```
dag_component_t
dag_component_get_subcomponent
(
   dag_component_t component,
   dag_component_code_t component_code,
int index
);
```

Return Value

The component requested or `NULL` if not found.

dag_component_get_subcomponent_count

Description

Retrieves the number of subcomponents of a given component.

```
int
dag_component_get_subcomponent_count
(
   dag_component_t component
);
```

Return Value

A count of the number of subcomponents.

dag_component_get_subcomponent_count_of_type

Description

Retrieves the number of components with a given component code.

```
int
dag_component_get_subcomponent_count
(
   dag_component_t component
);
```

Return Value

The number of components.

Chapter 8: Attribute Accessor Functions

Overview

All of the accessor functions defined in this chapter retrieve the value of an attribute. The only difference between them is the type of value they return. They are listed in alphabetical order.

Other functions relating to directly configuring the DAG card, modifiers, component and firmware are described in the previous and subsequent chapters of this programming guide.

The following designators are used in accessor functions:

Designator	Description
`card`	Refers to a DAG card.
`uuid`	An attribute identifier.
`component`	Refers to a component.

Functions

dag_config_get_boolean_attribute

```
uint8_t
dag_config_get_boolean_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid
);
```

dag_config_get_char_attribute

```
char
dag_config_get_char_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid
);
```

dag_config_get_int32_attribute

```
int32_t
dag_config_get_int32_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid
);
```

dag_config_get_int64_attribute

```
int64_t
dag_config_get_int64_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid
);
```

dag_config_get_string_attribute

```
const
char* dag_config_get_string_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid
);
```

dag_config_get_uint32_attribute

```
uint32_t
dag_config_get_uint32_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid
);
```

dag_config_get_uint64_attribute

```
uint64_t
dag_config_get_uint64_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid
);
```

Chapter 9: Modifier Functions

Overview

All of the modifier functions defined in this chapter assign a value to an attribute. The only difference between them is the type of value they return.

Other functions relating to directly configuring the DAG card, accessors, component and firmware are described in the previous and subsequent chapters of this programming guide.

The following designators are used in modifier functions:

Designator	Description
`card`	Refers to a DAG card.
`uuid`	An attribute identifier.
`value`	The value to assign to the attribute.

The following values are returned by modifier functions:

- `kDagErrInvalidCardRef` is returned if the card reference is invalid.
- `kDagErrNone` is returned on success.

Functions

dag_config_set_boolean_attribute

```
dag_err_t
dag_config_set_boolean_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid,
uint8_t value
);
```

dag_config_set_char_attribute

```
dag_err_t
dag_config_set_char_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid,
   char value
);
```

dag_config_set_int32_attribute

```
dag_err_t
dag_config_set_int32_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid,
   int32_t value
);
```

dag_config_set_int64_attribute

```
dag_err_t
```

```
dag_config_set_int64_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid,
   int64_t value
);
```

dag_config_set_null_attribute

```
dag_err_t
dag_config_set_null_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid
);
```

dag_config_set_string_attribute

```
dag_err_t
dag_config_set_string_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid,
   const char* value
);
```

dag_config_set_struct_attribute

```
dag_err_t dag_config_set_struct_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid,
   void* value
);
```

dag_config_set_uint32_attribute

```
dag_err_t
dag_config_set_uint32_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid,
   uint32_t value
);
```

dag_config_set_uint64_attribute

```
dag_err_t
dag_config_set_uint64_attribute
(
   dag_card_ref_t card,
   attr_uuid_t uuid,
   uint64_t value
)
```

Chapter 10: Firmware Functions

Overview

All of the firmware functions defined in this chapter load or read firmware on a card. The functions all return the same following function: `kDagErrNone`

The functions are listed in alphabetical order.

The following designators are used in modifier functions:

Designator	Description
`name`	The name of the device.
`card ref`	A valid pointer to a `dag_ref_t.`
`filename`	The name of the image to load.
`whch_pp`	The index starting from `0` of the packet processor to load.
`buffer`	A buffer to hold the SWID read from the card. It should be at least 128 bytes.
`length`	the size of the `buffer` in bytes.
`key`	The key to match the key in the ROM. If this key does not match, the Software ID (SWID) write will fail.

Functions

dag_firmware_load_pci

Description

Loads a PCI firmware image onto a card.

```
dag_err_t
dag_firmware_load_pci
(
   const char* name,
   dag_card_ref_t* card_ref,
   const char* filename
);
```

Note: card_ref must be a valid. This function will destroy the card and build the object again, including all components and attributes. Therefore any reference to the card prior to using this function will be in valid by the time the function returns. Any reference to a component or attribute will also be in valid by the time the function returns. After returning, `card_ref` will be a reference to a valid object.

Please refer to Chapter 14 earlier in this Programming Guide for more information on error codes.

dag_firmware_load_pp

Description

Loads an image onto one of the packet processors.

```
dag_err_t
dag_firmware_load_pp
(
   const char* name,
   dag_card_ref_t* card_ref,
   const char* filename,
   nt which_pp
);
```

Note: card_ref must be a valid. This function will destroy the card and build the object again, including all components and attributes. Therefore any reference to the card prior to using this function will be in valid by the time the function returns. Any reference to a component or attribute will also be in valid by the time the function returns. After returning, `card_ref` will be a reference to a valid object.

Please refer to Chapter 14 earlier in this Programming Guide for more information on error codes.

dag_firmware_read_swid

Description

Reads a Software ID (SWID) from the card.

```
dag_err_t
dag_firmware_read_swid
(
   dag_card_ref_t card,
   uint8_t* buffer,
   int length
);
```

dag_firmware_write_swid

Description

Writes a Software ID (SWID) to the card.

```
dag_err_t
dag_firmware_write_swid
(
   dag_card_ref_t card,
   uint8_t* buffer,
   int length,
   uint32_t key
);
```

Chapter 11: Data Structures and Constants

Overview

This chapter describes the types used by the functions and the enumerated types that you can use when setting or getting attribute values. They are listed in alphabetical order.

Types

attr_uuid_t

Description

A attribute identifier. This can be retrieved using the function:

```
dag_component_get_config_attribute_uuid
```

Include

dag_config.h

Attribute

???

connection_description_37t_t

Description

Used to describe the connection to be added to the DAG 3.7T card.

```
typedef struct
{

    connection_type37t_t mConnectionType;
    payload_type_t mPayloadType;
    direction_t mDirection;
    uint32_t mline;
    uint32_t mTimeslot;
    uint32_t mMask;
    uint32_t mConnectionNumber;

} connection_description_37t_t;
```

Include

```
dag_attributes_codes.h
```

Member	Type	Description
`mConnectionType`	connection_type37t_t	Describes the type of connection to be added i.e. single channel, hyper channel, sub channel of r.

connection_description_t

Description

A structure used to setup a connection on the DAG 7.1S card.

```
typedef struct
{
    uint8_t mTUG3_ID;
    uint8_t mVC_ID;
    uint8_t mTUG2_ID;
    uint8_t mTU_ID;
    uint8_t mPortNumber;
    connection_type_t mConnectionType;
    payload_type_t mPayloadType;
    uint8_t mScramble;
    uint8_t mHECCorrection;
    uint8_t mIdleCellMode;
    uint32_t mTimeslotMask;
} connection_description_t;
```

The outputs are described in the following table:

Output	Description
`mTUG_ID`	The TUG3 ID to use. Valid values re `0`,`1` and `2`. This field is only valid if the card is using E1.
`mVC_ID`	The VC ID to use. Valid values are `0` when the card line rate is configured for STM-1, and `0` to `3` when configured for STM-4 over E1. When configured for STM-1 over T1, valid values are `0` to `3` and when configured for STM-4 over T1, valid values are `0` to `11`.
`mTUG2_ID`	When using E1 or T1 valid values are `0` to `6`.
`mVC_ID`	The VC ID to use. Valid values are 0 when the card is configured for STM-1, and `0` to `3` when configured for STM-4 over E1. When configured for STM-1 over T1, valid values are `0` to `3` and when configured for STM-4 over T1, valid values are `0` to `11`.
`mTU_ID`	The TU ID to use. When configured for E1, valid values are `0` to `2` and when configured for T1 valid values are `0` to `3`.
`mPortNumber`	The DAG 7.1 S has 4 ports. Use this field to set the number of the port to con figure.
`mConnectionType`	The type of connection for which to configure the channel. See `connection_type_t` earlier in this chapter for valid values.
`mPayloadType`	The payload type to use for the connection. See `payload_type_t` later in this chapter for valid values.
`mScramble`	Disables or enable SONET frame scrambling on this connections.
`mHECCorrection`	Disables or enables HEC correction on this connection.
`mIdleCellMode`	Enable or disable idle cell mode. When enable idle cells will be dropped.
`mTimeslotMask`	A bitmask used to con figure the timeslots of the connection. The `mConnectType` field must be set to `kUseTimeslotConfig` to use this field.

connection_type_t

Descriiption

```
typedef enum
{
    kPCM31,
    kPCM30,
    kPCM24,
    kUseTimeslotConfig
} connection_type_t;
```

Include

```
dag_attribute_codes.h
```

crc_t

Description

Different CRC checking modes that the card can be configured to use.

```
typedef enum
{
    kCrcInvalid = -1,
    kCrcOff,
    kCrc16,
    kCrc32
} crc_t;
```

Include

```
dag_attribute_codes.h
```

dag71s_channelized_rev_id_t

Description

```
typedef enum
{
    kDag71sRevIdInvalid,
    kDag71sRevIdATM,
    kDag71sRevIdATMHDLC,
    kDag71sRevIdATMHDLCRAW,
    kDag71sRevIdHDLC,
    kDag71sRevIdHDLCRAW
} dag71s_channelized_rev_id_t;
```

Include

```
dag_attribute_codes.h
```

dag_attr_config_status_t

Description

```
typedef enum
{
    kDagAttrErr,
    kDagAttrStatus,
    kDagAttrConfig
} dag_attr_config_status_t;
```

Include

```
dag_attribute_codes.h
```

dag_card_ref_t

Description

A reference to a card. For example `dag_config` uses this type.

Include

```
dag_config.h
```

dag_card_t

Description

The type of DAG card

```
typedef enum
{
    kDagUnknown
    kDag35e,
    kDag35,
    kDag36d,
    kDag36e,
    kDag36ge,
    kDag37ge,
    kDag37t,
    kDag38,
    kDag42ge,
    kDag423ge,
    kDag42,
    kDag423,
    kDag43ge,
    kDag43s,
    kDag60,
    kDag61,
    kDag62,
    kDag70s,
    kDag70ge,
    kDag71s,

    kFirstDagCard = kDag35e,
    kLastDagCard = kDag71s

} dag_card_t;
```

Include

```
dag_config.h
```

dag_component_t

Description

A reference to a component. For example `dag_component_get_subcomponent` uses this type.

Include

```
dag_config.h
```

dag_err_t

Description

```
typedef enum
{
    kDagErrNone,
    kDagErrInvalidCardRef,
    kDagErrInvalidParameter,
    kDagErrNoSuchComponent,
    kDagErrNoSuchAttribute,
    kDagErrFirmwareVerifyFailed,
    kDagErrFirmwareLoadFailed,
    kDagErrSWIDerror,
    kDagErrSWIDInvalidBytes,
    kDagErrSWIDTimeout,
    kDagErrSWIDInvalidKey,
    kDagErrUnimplemented,
    kDagErrCardNotSupported
} dag_err_t;
```

The outputs are described in the following table:

Output	Description
`kDagErrNone`	No error occurred.
`kDagErrInvalidCardRef`	The card reference is invalid.
`kDagErrFirmwareLoadFailed`	Card failed to load the firmware image.
`kDagErrSWIDerror`	A general SWID related error occurred.
`kDagErrSWIDInvalidBytes`	An invalid number of bytes were given when reading/writing the SWID.
`kDagErrSWIDTimeout`	Timeout when communicating with the Xscale. Valid for the DAG 3.7T card.
`kDagErrSWIDInvalidKey`	The given key was in valid and did not match the one in ROM.

Include

```
dag_config.h
```

demapper_type_t

Description

Checks the type of Demapper on the DAG 3.7T card's firmware image.

```
typedef enum
{
    kDemapperTypeATM,
    kDemapperTypeHDLC
} demapper_type_t;
```

Include

```
dag_attribute_codes.h
```

erf_mux_37t_t

Description

To direct packet steering for the DAG 3.7T card.

```
typedef struct
{
   uint32_t mHost;
   uint32_t mLine;
   uint32_t mXscale;
} erf_mux_37t_t;
```

The packets can be steered in the directions defined in `erf_mux_steering_37t_t`.

```
typedef enum
{
    kErfToHost = 0x0,
    kErfToLine = 0x01,
    kErfToXscale = 0x02
} erf_mux_steering_37t_t;
```

Include

```
dag_attribute_codes_h
```

ethernet_mode_t

Description

Values for setting actual Ethernet modes available on a DAG card.

```
typedef enum
{
/**
10 Gigabit Ethernet (LAN).
*/
kEthernetMode10GBase_LR,

/**
*10 Gigabit Ethernet (WAN).
* Long wavelength (1310nm) single mode fiber.
*/
kEthernetMode10GBase_LW,

/**
* Short wavelength (850nm) Multimode fiber with 66B encoding.
*/
kEthernetMode10GBase_SR,
/**
* Extra long wavelength (1550nm) single mode fiber with 66B encoding.
*/
kEthernetMode10GBase_ER

} ethernet_mode_t;
```

Include

```
dag_attribute_codes.h
```

led_status_t

Description

The status of the LED on the DAG 3.7T Pod. Use with the attribute `kUint32AttributeLEDStatus` to change properties of an LED on the pod.

```
typedef enum
{
    kLEDOn
    kLEDOff
    kLEDAtBlinkRate0,
} led_status_t;
```

Include

```
dag_attribute_codes.h
```

line_rate_t

Description

Line rates for which the cards can be configured.

```
typedef enum
{
    kLineRateAuto,
    kLineRateOC3c,
    kLineRateOC12c,
    kLineRateOC48c,
    kLineRateOC192c,
    kLineRateEthernet10,
    kLineRateEthernet100,
    kLineRateEthernet1000

} line_rate_t;
```

Include

```
dag_attribute_codes.h
```

line_type_t

Description

An enumerated type denoting the various line types of the DAG 3.7T and DAG7.1s cards. For use with the attribute `kUint32AttributeLineType`.

```
typedef enum
{
    kLineTypeOff,
    kLineTypeE1,
    kLineTypeE1crc,
    kLineTypeE1unframed,
    kLineTypeT1,
    kLineTypeT1sf,
    kLineTypeT1esf
} line_type_t;
```

Include

```
dag_attribute_codes.h
```

master_slave_t

Description

Configures the card in master or slave mode.

```
typedef enum
{
    kMasterSlaveInvalid,
    kMaster,
    kSlave
} muster_slave_t;
```

Include

```
dag_attribute_codes.h
```

mux_t

Description

Configures the MUX on a DAG 3.7 GP/GF

```
typedef enum
{
    kMuxMerge,
    kMuxSplit
} mux_t;
```

Include

```
dag_attribute_codes.h
```

network_mode_t

Description

Sets the network mode.

```
typedef enum
{
    kNetworkModeInvalid,
    kNetworkModeATM,
    kNetworkModePoS,
    kNetworkModeRAW,
    kNetworkModeEth
} network_mode_t;
```

Include

```
dag_attribute_codes.h
```

payload_mapping_t

Description

Defines the payload mapping type. Used with the attribute `kUint32AttributePayloadMapping`.

```
typedef enum
{
    kPayloadMappingDisabled,
    kPayloadMappingAsync,
    kPayloadMappingBitSync,
    kPayloadMappingByteSync1,
    kPayloadMappingByteSync2
} payloadmapping_t
```

Include

```
dag_attribute_codes.h
```

payload_type_t

Description

```
typedef enum
{
    kPayloadTypeNotConfigured,
    kPayloadTypeATM
    kPayloadTypeHDLC,
    kPayloadTypeRAW
} payload_type_t;
```

Include

```
dag_attribute_codes.h
```

pci_bus_speed_t

Description

Speeds of the PCI bus. This can be detected using the `pbm` component.

```
typedef enum {
    kPCIBusSpeed33Mhz,
    kPCIBusSpeed66Mhz,
    kPCIBusSpeed100Mhz,
    kPCIBusSpeed133Mhz,
    kPCIBusSpeedUnknown,
    kPCIBusSpeedUnstable
} pci_bus_speed_t;
```

Include

```
dag_attribute_codes.h
```

sonet_type_t

Description

```
typedef enum
{
    kSonetTypeInvalid,
    kSonetTypeChannelized,
    kSonetTypeConcatenated
} sonet_type_t;
```

Include

```
dag_attribute_codes.h
```

steer_t

Description

```
typedef enum
{
   kSteerStream0,
   kSteerParity,
   kSteerCrc,
   kSteerIface
}
```

Include

```
dag_attribute_codes.h
```

terf_strip_t

Description

Used to set the CRC stripping functionality on cards with the TERF component, `kComponentTerf`. The TERF component requires transmit firmware to be installed.

```
typedef enum
{
    kTerfStripInvalid,
    kTerfNoStrip,
    kTerfStrip16,
    kTerfStrip32
} terf_strip_t;
```

Include

```
dag_attribute_codes.h
```

termination_t

Description

The termination mode.

```
typedef enum
{
    /* Both external. */
    kTerminationExternal,

    /* One internal, one external. */
    kTerminationRxExternalTx75ohm,
    kTerminationRxExternalTx100ohm,
    kTerminationRxExternalTx120ohm,
    kTerminationRx75ohmTxExternal,
    kTerminationRx100ohmTxExternal,
    kTerminationRx120ohmTxExternal,

    /* Both internal. */
    kTermination75ohm,
    kTermination100ohm,
    kTermination120ohm

} termination_t;
```

Include

```
dag_attribute_codes.h
```

tributary_unit_t

Description

Sets the tributary unit on the DAG7.1S card which is currently the only card that supports the `tributary_unit_t`.

```
typedef enum
{
    kTU11,
    kTU12
} tributary_unit_t;
```

Include

```
dag_attribute_codes.h
```

vc_pointer_state_t

Description

Different pointer states of virtual containers

```
 typedef enum
{
    kLossOfPointer,
    kAlarmSignalIndicator,
    kPointerValid,
    kConcatenationIndicator
} vc_pointer_state_t;
```

Include

```
dag_attribute_codes.h
```

vc_size_t

Description

Different virtual container sizes that the cards can be configured to.

```
typedef enum
{
   kVC3,
   kVC4,
   kVC4C
} vc_size_t;
```

Include

```
dag_attribute_codes.h
```

zero_code_suppress_t

Description

```
typedef enum
{
    kZeroCodeSuppressB8ZS,
    kZeroCodeSuppressAMI

} zero_code_suppress_t;
```

Include

```
dag_attribute_codes.h
```

Version History

Version	Date	Reason
1-3		Old Versions.
4	May 2006	Inclusion of DAG 7.1S.
5	August 2006	Inclusion of DAG 8.2X.
6	September 2007	Inclusion of DAG 5.0SG2, DAG 5.2X, DAG 5.2SXA.
6.1	November 2007	Correction to TOC
7	November 2008	Updated for Dag Software release 3.2.1.