endace
accelerated

IXP Filtering Guide

EDM04-11

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

Whilst every effort has been made to ensure accuracy, neither Endace Technology Limited nor any employee of the company, shall be liable on any ground whatsoever to any party in respect of decisions or actions they may make as a result of using this information.

Endace Technology Limited has taken great effort to verify the accuracy of this manual, but nothing herein should be construed as a warranty and Endace shall not be liable for technical or editorial errors or omissions contained herein.

In accordance with the Endace Technology Limited policy of continuing development, the information contained herein is subject to change without notice.

Website

http://www.endace.com

Copyright 2008 Endace Technology Ltd. All rights reserved.

Contents

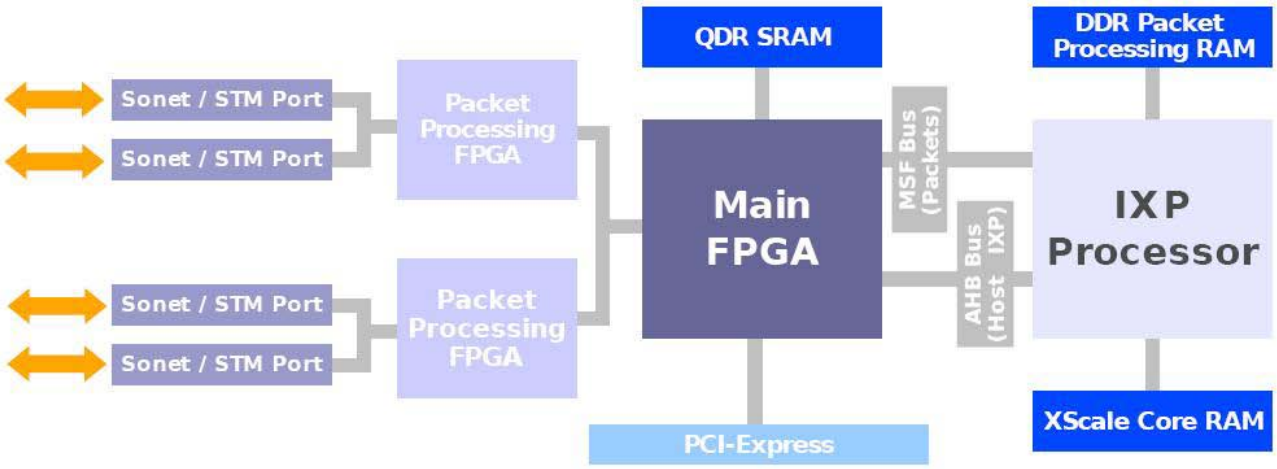
IXP Filtering Introduction

IXP Filtering Overview

The IXP Filter is a collection of host and embedded software that performs Internet Protocol (IPv4/IPv6) and layer 3 filtering on PoS packets. The IXP Filtering is specific to DAG 7.1S cards, all the filtering is done internally on the card. A host API library is supplied to configure and initiate the filtering, however no host process is required to maintain the filtering once configured and running.

DAG 7.1S Card Overview

The Endace DAG 7.1S is a four port SONET/STM capture card that can support data rates of 4 × oc3 or 2 × oc12. To provide the extra functionality supported by the card (in this case IP filtering) a Intel IXP network processor has been added. The diagram below illustrates the main components of a DAG 7.1S card.

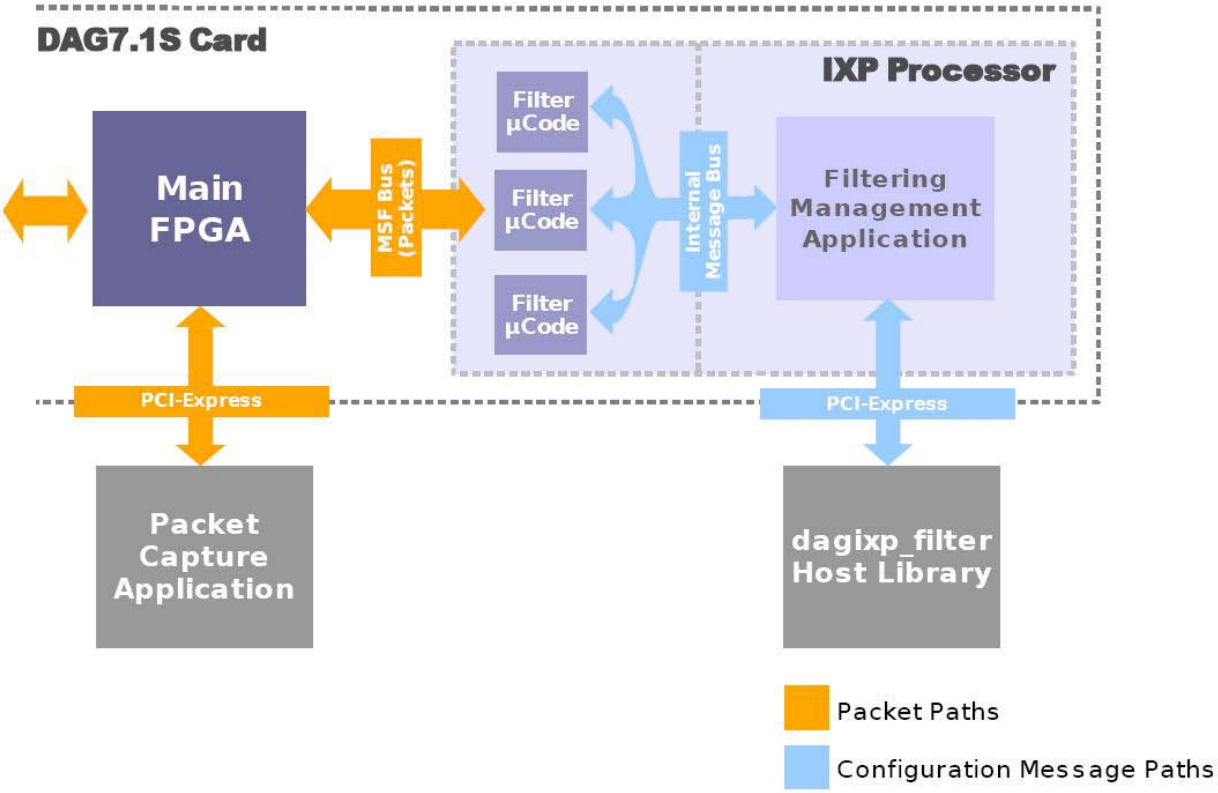


By default when a DAG 7.1S card is reset (or powered up) the main FPGA is configured to route packets from the line to the host and from the host to the line, this is standard DAG card behavior. In this way the DAG 7.1S can be used as a standard Endace network capture card without IP filtering, however when IP filtering is required, packets should be routed to and from the IXP.

Packet routing within the card is configurable, allowing packets to be routed from anyone of the three sources (line, host or IXP) to any other, including back to themselves. This routing is fully user configurable. See the *IXP Filtering API and Packet Routing* (page 8) for more information on how to achieve the correct routing.

IXP Filter Software Overview

The IXP filter software consists of a host library that interfaces with a DAG 7.1S card, an embedded software management application that executes on the card and a collection of targeted packet processing programs. The diagram below illustrates the logical layout of the different software components.



The `dagixp_filter` Host Library provides an API by which filter rules can be constructed and compiled into user defined ruleset. The host library also provides the functionality to load ruleset into the card and query the current filtering status. The `dagixp_filter_loader` application uses this library to load the rulesets contained in a configuration file to the card.

The Filtering Management Application is an embedded program that runs inside the IXP network processor and manages the Filtering microcode (µCode) programmes. It is also the program that the host library communicates with when loading the filter rulesets and querying the filtering status.

The Filter Microcode (µCode) performs the packet filtering. It accepts the packet records from the main FPGA and applies the filtering rules. Packets may be dropped by the microcode or routed back out the line or to the host depending on the rules installed.

All configuration of the IXP filter is done via the `dagixp_filter` Host Library, rom image files are provided that contain the embedded software for both the Filtering Management Application and the Filter MicroCode.

Coloured Packet Records

As packets pass through the filtering software the packet record is modified to change it's ERF type and to add a color field in the ERF record header (the length of the record might also be altered based on the snap length of the rules). The color field is a 14-bit wide value that will match the user defined tag of the rule that the packet hits, refer to *EDM11-01 ERF types* for more information.

IXP Filter Rulesets

The IXP filter can filter on Internet Protocol (IP) version 4 and version 6 packets encapsulated in PoS frames.

Rulesets

The IXP filter accepts a collection of rules packaged up within a logical ruleset, multiple rulesets can be loaded into the card but only one can be active at a time. A rule contains a set of fields that is compared with the packet, if all the fields of the rule match then the rule is said to hit, if one of more of the fields don't match the rule is said to miss. If the rule hits the action and steering attributes of the rule determines whether the packet is dropped or routed to the host/line. If none of the rules produce a hit then the packet is dropped.

Rules are priority ordered within a ruleset, rules with the highest priority are compared with the packet first, if a rule hits the rest of the rules are ignored. A rule can target either IPv4 or IPv6 packets though it is not possible to have a rule that targets both. If no ruleset has been activated the filtering software goes into loopback mode. See the Modes of Operation section on page 5 for more information on filtering modes.

Internet Protocol Version 4 Rules

An IPv4 rule will match only if the layer 2 type of the packet being compared is IPv4, any other sort of layer 2 type (for example IPv6, ARP or IPX) will result in an automatic rule miss. The following table lists IPv4 header fields that a rule can apply a bit masked filter to, multiple fields may be defined in a single rule.

IPv4 Header Field	Description
IP Source Address	A 32-bit bit-masked filter may be applied to the IP source address , if the filter doesn't match the rule misses.
IP Destination Address	A 32-bit bit-masked filter may be applied to the IP destination address , if the filter doesn't match the rule misses.

A rule can also filter on the IP protocol field of a packet however this is not a bit masked type filter, instead a direct value match is performed on the field. If the IP protocol field is set to filter on TCP, UDP or SCTP type packets, up to 254 additional source and destination port filters can be added to the rule. See the TCP, UDP and SCTP Port Filtering section on page 4 for more information. ICMP packets can also be filtered in a similar way, with up to 254 ICMP type filters. See the ICMP Type Filtering section on page 4 for more information.

IP options are automatically skipped by the filtering process, however filter rules cannot target fields within the IP options.

Internet Protocol Version 6 Rules

IPv6 rules target packets with an IPv6 layer 2 type only, any other packet types automatically result in a rule miss. The following list IPv6 header fields that a rule can apply a bit masked filter to, multiple fields may be defined in a single rule.

IPv6 Header Field	Description
IP Source Address	A 128-bit bit-masked filter may be applied to the IP source address , if the filter doesn't match the rule misses.
IP Destination Address	A 128-bit bit-masked filter may be applied to the IP destination address , if the filter doesn't match the rule misses.
Flow Label	A 20-bit bit-masked filter may be applied to the flow label field of an IPv6 header, if the filter doesn't match the rule misses.

As with an IPv4 rule, layer 3 protocols (TCP, UDP, SCTP or ICMP) can be filtered on. See the TCP, UDP and SCTP Port Filtering section and the ICMP Type Filtering section on page 4 for more information.

IPv6 extension headers are automatically skipped by the filtering process, the following table illustrates which extension headers are supported, if an unknown extension header is found, the packet is dropped and a statistics counter is incremented. Currently filter rules can't target fields within extension headers.

Name	Supported	Description
Hop-by-Hop Options Header	yes	Contains options for hop-by-hop processing.
Routing Header	yes	Contains routing information for the packet.
Fragment Header	yes	Contains information for fragmenting and reassembling the packet.
Authentication Header	yes	Used for packet authentication.
Encapsulated Security Payload Header	no	Provides fields for packet encryption and security.
Destination Options Header	yes	Contains optional information for the destination node of the packet.

TCP, UDP and SCTP Port Filtering

As well as IP header filtering, a rule can filter on TCP, UDP or SCTP source and destination port numbers. Two types of port filters can be added to a rule; bit-masked or port range, bit-masked filters take a value and mask pair to compare against the packet, a port range filter takes a maximum and minimum port value to compare with the packet. Up to 254 source and destination port ranges filters can be added to a rule or a single bit-masked filter. If no port filters are added to the rule the port value in the packet is ignored by the rule. If multiple port filters are added to the rule, each is compared with the packet value and OR'ed together to produce the result, for example if a rule has two source port ranges 0-25 and 80-90 as well as two destination port ranges 25-50 and 110-120, then only packets with a source port value of between 0 and 25 or 80 and 90 and with a destination port value of between 25-50 or 110-120 will result in a rule hit.

The maximum and minimum values of a port range are inclusive values. If wanting to filter on a single port value, a range can be added to the rule with both the minimum and maximum values set to the same value.

ICMP Type Filtering

ICMP packets can be filtered on their type fields in much the same way as ports can be filtered on for TCP, UDP and SCTP packets. The only differences between port filter and ICMP type filtering is that the type values are only 8-bit rather than the 16-bit for port values and there is a single ICMP type filter list rather than the two source and destination port filter lists.

PoS Frame Header Formats

Both PPP with HDLC (RFC 1662) and Cisco type PoS encapsulation is supported, 16 or 32 bit CRCs are also supported. The following table lists the PoS header formats supported by the filtering software.

PoS Header Values	Format	Layer 2 Type
0xFF030021	RFC 1662	IPv4
0x0F000800	Cisco	IPv4
0xFF030057	RFC 1662	IPv6
0x0F0086DD	Cisco	IPv6
0xFF030281	RFC 1662	IP with Unicast MPLS shims
0xFF030283	RFC 1662	IP with Multicast MPLS shims
0x0F008847	Cisco	IP with Unicast MPLS shims
0x0F008848	Cisco	IIP with Multicast MPLS shims

Multi-Protocol Label Switching (MPLS) Support

The filtering microcode can handle packets with up to 10 MPLS shim headers inserted between the HDLC and IP headers. If more than 10 shims are present, the packet is automatically dropped and a statistic counter updated. Currently filter rules cannot target fields within MPLS shims.

Modes of Operation

The filtering software has two possible modes of operation, the first is filter mode, where packets that are presented to the IXP network processor are filtered based on the activated ruleset, this is what is described above.

Loopback Mode

The second mode of operation is loopback mode, where incoming packets are looped back out of the IXP without being filtered. In loopback mode the ERF type of the packet record is still modified to have a color field, however the value in the color field is always 0x3FFF, this is used to distinguish loopback packets from packets that have been filtered.

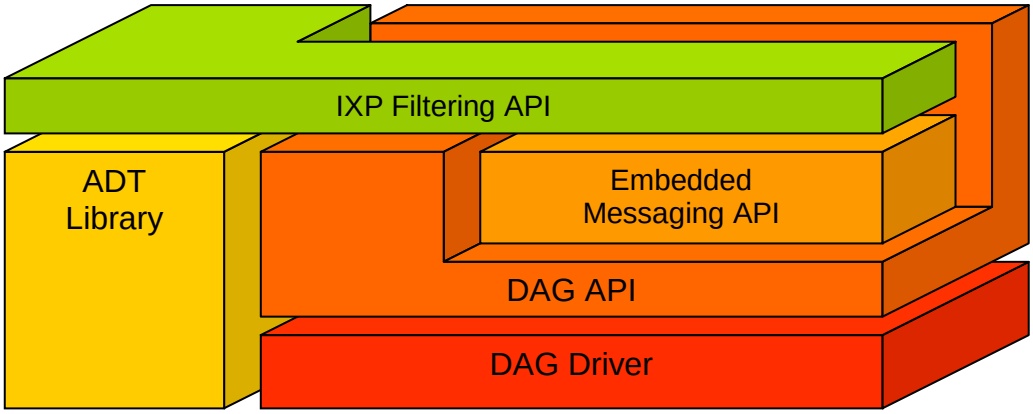
Loopback mode is not directly controllable by the user, instead it is automatically entered when the filtering software

- is initially started and no rulesets have been activated.
- is activating a new ruleset (the process of deactivating the old ruleset and activating the new ruleset is not instantaneous, during this time the software is put in looopback mode so packets are not lost).

IXP Filtering API Overview

IXP Filtering API Dependencies

Because the IXP Filtering API reads and writes configuration information to from the IXP network processor on board the DAG card, it requires the DAG driver to be running and it expects both the DAG API and Embedded Messaging API (dagema) libraries to be present. The following figure illustrates the logical layering of the various libraries required by the IXP Filtering API.



IXP Filtering API Structure

The DSM API is divided into six logical sections which all are contained within a single library file.

Name	Description
Initialisation and Termination	Provides two functions to initialise the library prior to use and clean up when finished with the library.
Rule Construction	Provides the functionality to create and modify rulesets as well as individual IPv4 and IPv6 rules.
Rule Attributes	Contains functions to modify the non-filtering attributes of a particular rule.
Ruleset Loading	Contains the necessary functionality to load a ruleset into a DAG 7.1S card and activate it.
Statistics	Provides the functionality to retrieve the current statistics associated with the filtering process from the DAG card.
Utilities	Provides miscellaneous utilities for the IXP Filtering API.

IXP Filtering API and the Embedded Messaging API

The IXP Filtering API relies on the Embedded Messaging API (EMA) to provide a connection to the IXP network processor on the DAG card. Therefore an open EMA connection should be established prior to loading/activating a ruleset on the card, the following code snippet demonstrates typical usage.
Note: that error checking has been removed for brevity.

```
/* create a ruleset */
ruleset_h = dagixp_filter_create_ruleset();

/* populate the ruleset with rules */
...
/* open the dag card at location 0 */
dag_parse_name (“dag0”,  dagname, DAGNAME_BUFSIZE, &stream);
dagfd = dag_open (dagname);

/* reset the IXP network processor on the card to put it in a known state */
/* (this is not necessary but recommended if configuring for the first time) */
dagema_reset_processor (dagfd, 0);

/* open a connection to the processor */
dagema_open_conn (dagfd);

/* download the ruleset to the card */
dagixp_filter_download_ruleset (dagfd, kAllInterfaces, ruleset_h);

/* activate the ruleset */
dagixp_filter_activate_ruleset (dagfd, kAllInterfaces, ruleset_h);

/* delete the ruleset */
dagixp_filter_delete_ruleset (ruleset_h);

/* close the connection */
dagema_close_conn (dagfd, 0);

/* close the dag card */
dag_close (dagfd);
```

Refer to *EDM04-15 Embedded Messaging API* document for more information on opening and closing connections as well as resetting the IXP network processor.

Note: After resetting the IXP, connections should be added or removed one at a time (repeat as many times as required for multiple connections). Do not 'reset' the IXP each time as this will likely result in failure to receive some packets.

IXP Filtering API and Packet Routing

For the IP filtering to work, it requires packet records to be past to the IXP network processor, by default packets received from the line are routed directly to the host bypassing the IXP, and vise versa for packets received from the host.

The DAG Configuration and Status API library has a set of functions that control the routing of packet records within DAG 7.1S cards, the following code snippet shows how the packet steering would typically be setup.

```
char            dagname[DAGNAME_BUFSIZE];
int             dagstream;
dag_card_ref_t  card_ref;
dag_component_t root_component;
dag_component_t erfmux;
attr_uuid_t     line_steering_attr;
attr_uuid_t     ixp_steering_attr;
attr_uuid_t     host_steering_attr;

/* open the dag card at location 0 */
dag_parse_name (“dag0”, dagname, DAGNAME_BUFSIZE, &stream);
/* open a reference to the card and get the root component */
card_ref = dag_config_init (dagname);
root_component = dag_config_get_root_component (card_ref);

/* get the ERF MUS component */
erfmux = dag_component_get_subcomponent (root_component, kComponentErfMux, 0);

/* get the three steering attributes */
line_steering_attr
       = dag_component_get_attribute_uuid (erfmux,
kUint32AttributeLineSteeringMode);
ixp_steering_attr
       = dag_component_get_attribute_uuid (erfmux,
kUint32AttributeIXPSteeringMode);
host_steering_attr
       = dag_component_get_attribute_uuid (erfmux,
kUint32AttributeHostSteeringMode);

/* steer the packets from the line to the IXP */
dag_config_set_uint32_attribute (card_ref, line_steering_attr, kSteerIXP);
/* use the direction bits in the packet record header to determine the steering
*/

dag_config_set_uint32_attribute (card_ref, ixp_steering_attr,
kSteerDirectionBit);

/* steer packets from the host to the IXP */
dag_config_set_uint32_attribute (card_ref, host_steering_attr, kSteerIXP);
/* clean up */
dag_config_dispose (card_ref);
```

In the above example, packets are routed from the line to the IXP, from the host to the IXP and from the IXP to either the host or the line depending on the direction bit in the packet record header. The direction bit is modified when IP filtering is enabled and the value of the bit is user configurable per rule. For more information on how to configure the packet routing refer to the *EDM04-08 DAG Configuration and Status APIP Programming Guide.*

IXP Filtering API Typical Usage

The following steps are usually taken to configure the card:

1. Configure the card for receiving traffic using the DAG Configuration and Status API or the command line dagconfig program.
2. Create a new ruleset and populate it with filtering rules using the IXP filtering API.
3. Open an EMA connection to the IXP network processor on the DAG card, if this is the first time a connection is opened the IXP processor should be reset first.
4. Configure the packet routing module on the card using the Configuration and Status API, to route packets from the line to the IXP.
5. Clean any left over ruleset from the card and then load the new ruleset into the card and activate it.
6. Clean up the ruleset and close the connection to the EMA and DAG card.

IXP Filtering API and Multiple Threads

The IXP Filtering API library is not thread safe, users are required to wrap function calls, were appropriate, with their own thread safe mechanism (for example semaphores or mutexes).

Startup and Shutdown

The two functions contained in this section are required to initialize and tear down the IXP filtering library. The startup function must be called prior to using the library, failure to do so will result in unpredictable behavior.

dagixp_filter_startup Function

Purpose Initialises the use of the DAG IXP Filter library.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_startup (void)
```

Parameters This function has no parameters.

Returns `0` if the library was initialised and `-1` if an error occurred. Call `dagixp_filter_get_last_error` to retrieve the error code.

EALREADYOPEN（this function has been called previously）

ENOMEM (not enough free memory available)

Comments This `dagixp_filter_startup` function must be the first IXP filter API function called by an application or library. The application or library can only issue further IXP filtering API function calls after successfully calling `dagixp_filter_startup.`

The `dagixp_filter_startup` function can be called multiple times, but on subsequent calls `-1` will be returned and `dagixp_filter_get_last_error` will return EALREADYOPEN.

When it has finished using the services of the IXP filtering library, the application must call `dagixp_filter_shutdown` to allow the DAG IXP Filter library to free any resources allocated for the filtering process.

dagixp_filter_shutdown Function

Purpose Terminates use of the DAG IXP Filter library.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_shutdown (void)
```

Parameters This function has no parameters.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (library not initialised)

Comments An application is required to perform a successful `dagixp_filter_startup` call before it can use the IXP filtering API library services. When it has completed the use of the IXP filtering library, the application must call `dagixp_filter_shutdown` to free any resources allocated by the IXP filtering library on behalf of the application.

There is no internal reference count for `dagixp_filter_startup` and `dagixp_filter_shutdown` functions, therefore `dagixp_filter_shutdown` will free all resources regardless of the number of calls made to `dagixp_filter_startup.`

The `dagixp_filter_shutdown` function doesn't delete any rulesets that have been created by the application, it is the callers responsibility to delete the any rulesets prior to calling `dagixp_filter_shutdown.`

Filter Rule Construction

Functions contained in this section provide the functionality to create a set of filters rules that can be loaded into a DAG 7.1S card.

dagixp_filter_create_ruleset Function

Purpose Creates a new empty ruleset.

Declared In `dagixp_filter.h`

Prototype
```
RulesetH dagixp_filter_create_ruleset (void)
```

Parameters This function has no parameters.

Returns A handle to a new ruleset is returned if successful. Otherwise, a NULL value is returned to indicate an error, and a specific error number can be retrieved by calling `dagdsm_get_last_error`.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

ENOMEM (memory allocation error)

Comments The `dagixp_filter_create_ruleset` function creates a new ruleset and returns a handle that can be used in subsequent IXP filtering library calls that require a ruleset handle.

When finished with the ruleset, `dagixp_filter_delete_ruleset` must be called to free resources allocated to the ruleset. Rulesets are not automatically deleted when a the IXP filtering library is shutdown (by calling `dagixp_filter_shutdown`), it is the callers responsibility to ensure all rulesets are deleted, prior to the application terminating, otherwise memory leaks will occur.

Rulesets are created empty, no filter rules are defined within the ruleset, an empty ruleset cannot be loaded into a DAG 7.1S card.

dagixp_filter_delete_ruleset Function

Purpose Deletes an existing ruleset.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_delete_ruleset (RulesetH ruleset_h)
```

Parameters → `ruleset_h`

A handle to the ruleset to delete.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (library not initialised)

EINVAL (invalid parameter)

Comments `dagixp_filter_delete_ruleset` invalidates the specific ruleset handle, frees the resources allocated and deletes all the filter rules contained in the ruleset.

If the `dagixp_filter_delete_ruleset` call was successful, the `ruleset_h` handle should be discarded, unpredictable behaviour will result if the ruleset handle is continued to be used.

There is no need to free all the individual rules of a ruleset prior to deleting it, this function will clean up any rules that remain in the ruleset. For this reason individual rule handles associated with a deleted ruleset should be discarded, unpredictable behaviour will result if the rule handle is continued to be used.

dagixp_filter_empty_ruleset Function

Purpose Removes all the filter rules from a ruleset.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_empty_ruleset (RulesetH ruleset_h)
```

Parameters → `ruleset_h`
A handle to the ruleset to remove all the rules from.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.
ENOTOPEN (library not initialised)
EINVAL (invalid parameter)

Comments This function removes all the rules from a ruleset and frees all the resources allocated for them. Any rule handles created for the ruleset should be discarded, continuing to use them will result in unpredictable behaviour.

dagixp_filter_ruleset_rule_count Function

Purpose Returns the number of rules in a ruleset.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_ruleset_rule_count (RulesetH ruleset_h)
```

Parameters → `ruleset_h`
A handle to the ruleset to get the rule count for.

Returns The return value is a positive number indicating the number of rules in the ruleset if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.
ENOTOPEN (library not initialised)
EINVAL (invalid parameter)

dagixp_filter_ruleset_get_rule_at Function

Purpose Returns a handle to a rule at a given index inside a ruleset.

Declared In `dagixp_filter.h`

Prototype
```
RuleH dagixp_filter_ruleset_get_rule_at (RulesetH ruleset_h,
uint32_t index)
```

Parameters → `ruleset_h`

A handle to the ruleset to get the rule from.

→ `index`

Zero based index of the rule to retrieve.

Returns A handle to the rule at the given index is returned if successful. Otherwise, a NULL value is returned to indicate an error, and a specific error number can be retrieved by calling `dagdsm_get_last_error.`

ENOTOPEN (library not initialised)

EINVAL (invalid parameter)

Comments Rules are stored inside the ruleset in priority order, this is the same order that is used by the filtering module to determine the best filter hit. Rules with higher priority appear earlier in the ruleset and therefore have a lower index number. Because of this, when rules are added to a ruleset the index number of the rules may be rearranged based on the priority of the new rule. It should not be assumed that the last rule that has been added is at the last index, unless it is guaranteed that the rule has the lowest priority.

Calling `dagixp_filter_set_rule_priority` will also change the ordering of the rules inside the ruleset and therefore the indices of the rules. Use `dagixp_filter_set_rule_priority` with caution when iterating through rules inside a ruleset.

dagixp_filter_remove_rule Function

Purpose Deletes a rule from a ruleset.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_remove_rule (RulesetH ruleset_h, RuleH rule_h)
```

Parameters → `ruleset_h`

A handle to the ruleset to remove the rule from.

→ `rule_h`

Handle to the rule to remove from the ruleset.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (library not initialised)

EINVAL (invalid parameter)

ENOMEM (memory error)

Comments When a rule is removed from a ruleset, the memory allocated for it is freed and the ruleset is updated. If `dagixp_filter_remove_rule` is successful the rule handle should be discarded, continuing to use the handle will result in unpredictable behaviour.

The rule being removed must belong to the given ruleset, this function will fail with an error code of EINVAL if the rule doesn't belong to the ruleset.

dagixp_filter_create_ipv4_rule Function

Purpose Creates a new rule targeted at Internet Protocol version 4 packets and adds it to the given ruleset.

Declared In `dagixp_filter.h`

Prototype
```
RuleH dagixp_filter_create_ipv4_rule (RulesetH ruleset_h, action_t
action, uint16_t rule_tag, uint16_t priority)
```

Parameters

→ `ruleset_h`

A handle to the ruleset to add the new rule to.

→ `action`

The action to assign to the rule, this can be one of the following constants.

`kReject:` If the rule hits the packet is rejected (dropped).

`kAccept:` If the rule hits the packet is accepted, and routed to either the host or the line based on the rule steering attribute and card configuration.

→ `rule_tag`

User defined decimal number that is inserted into the packet record if the rule hits and the action is kAccept. The upper two bits of this value are ignored, the range of possible values are 0-4095. Multiple rules can have the same tag number.

→ `priority`

Decimal number that defines the priority of the rule, the lower the number the higher the priority, a zero value indicates the rule has the highest priority.

Returns A handle to a new rule is returned if successful. Otherwise, a NULL value is returned to indicate an error, and a specific error number can be retrieved by calling `dagdsm_get_last_error.`

Possible error codes:

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

ENOMEM (memory allocation error)

EINVAL (invalid parameter)

Comments A successful call to `dagixp_filter_create_ipv4_rule` will create a blank filter and add it to the given ruleset. The new rule will only target IPv4 packets, any other packets types will automatically be skipped by the rule.

The following table illustrates the default values for the rule parameters.

Rule Filter Fields	Default Value	
	Comparand	Mask
Source Address	0.0.0.0	0.0.0.0
Destination Address	0.0.0.0	0.0.0.0
IP Protocol	0	0
Source Port(s)	0	0
Desination Port(s)	0	0
IMCP Type(s)	0	0
Rule Attributes	**Default Value**	
Rule Priority	Value set by priority argument	
Rule Action	Value set by action argument	
Rule Snap Length	65528	
Rule Steering	Steer packets to the host	
Rule Tag	Value set by rule_tag argument	

Note: The Source Port(s) and Destiantion Port(s) fields are only used if the IP Protocol field is set to 6 (TCP), 17 (UDP) or 132 (SCTP). The IMCP Type(s) field is only used if the IP Protocol field is set to 1 (ICMP).

As shown in the above table the newly created rule will always hit IPv4 packets, because all the rule parameters are zero.

If a rule hits on a packet, the action parameter determines what should be done with it, the only two possible values are kReject or kAccept.

The lower the priority number the higher the priority of the rule, zero has the highest priority and 65534 has the lowest priority. Although multiple rules may be defined with the same priority, it is not recommended, because rules with the same priority are compared with the packet in random order, it is not possible to determine the exact order of these rules.

The `rule_tag` is a user defined 14-bit value that is copied into the lctr/color field of the ERF packet record when a rule hits and the rule action is set to accept the packet. Refer to *EDM11-01 Endace Extensible Record Format* for the format of a filtered packet record.

dagixp_filter_create_ipv6_rule Function

Purpose Creates a new rule targeted at Internet Protocol version 6 packets and adds it to the given ruleset.

Declared In `dagixp_filter.h`

Prototype
```
RuleH dagixp_filter_create_ipv6_rule (RulesetH ruleset_h, action_t
action, uint16_t rule_tag, uint16_t priority)
```

Parameters → `ruleset_h`

A handle to the ruleset to add the new rule to.

→ `action`

The action to assign to the rule, this can be one of the following constants.

`kReject:` If the rule hits the packet is rejected (dropped).

`kAccept:` If the rule hits the packet is accepted, and routed to either the host or the line based on the rule steering attribute and card configuration.

→ `rule_tag`

User defined decimal number that is inserted into the packet record if the rule hits and the action is kAccept. The upper two bits of this value are ignored, the range of possible values are 0-4095. Multiple rules can have the same tag number.

→ `priority`

Decimal number that defines the priority of the rule, the lower the number the higher the priority, a zero value indicates the rule has the highest priority.

Returns A handle to a new rule is returned if successful. Otherwise, a NULL value is returned to indicate an error, and a specific error number can be retrieved by calling `dagdsm_get_last_error.`

Possible error codes:

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

ENOMEM (memory allocation error)

EINVAL (invalid parameter)

Comments A successful call to `dagixp_filter_create_ipv6_rule` will create a blank filter and add it to the given ruleset. The new rule will only target IPv6 packets, any other packets types will automatically be skipped by the rule.

The following table illustrates the default values for the rule parameters.

Rule Filter Fields	Default Value	
	Comparand	Mask
Source Address	0:0:0:0:0:0:0:0	0:0:0:0:0:0:0:0
Destination Address	0:0:0:0:0:0:0:0	0:0:0:0:0:0:0:0
IP Protocol	0	0
Flow Label	0	0
Source Port(s)	0	0
Destination Port(s)	0	0
ICMP Type(s)	0	0

Rule Meta Data	Default Value
Rule Priority	Value set by priority argument
Rule Action	Value set by action argument
Rule Snap Length	65528
Rule Steering	Steer the packet to the host
Rule Tag	Value set by rule_tag argument

Note: The Source Port(s) and Destiantion Port(s) fields are only used if the IP Protocol field is set to 6 (TCP), 17 (UDP) or 132 (SCTP). The IMCP Type(s) field is only used if the IP Protocol field is set to 1 (ICMP). See the comments in the dagixp_filter_create_ipv4_rule Function section on page 15 for more information on creating a new rule.

dagixp_filter_set_ipv4_source_field Function

Purpose Sets the IPv4 source address to filter on.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_set_ipv4_source_field (RuleH rule_h, struct
in_addr *src, struct in_addr *mask)
```

Parameters → `rule_h`

Handle to an IPv4 rule. The rule should have been created by `the dagixp_filter_create_ipv4_rule function.`

→ `src`

Pointer to an `in_addr` structure that represents the source address to use as the comparand.

→ `mask`

Pointer to an `in_addr` structure that represents the source address mask to use for the filter.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_get_ipv4_source_field Function

Purpose Gets the IPv4 source address used by the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_ipv4_source_field (RuleH rule_h, struct
in_addr *src, struct in_addr *mask)
```

Parameters → `rule_h`

Handle to an IPv4 rule. The rule should have been created by the `dagixp_filter_create_ipv4_rule` function.

← `src`

Pointer to an `in_addr` structure that receives the comparand part of the source address field.

← `mask`

Pointer to an `in_addr` structure that receives the mask part of the source address field.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_set_ipv4_dest_field Function

Purpose Sets the IPv4 destination address to filter on.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_set_ipv4_dest_field (RuleH rule_h, struct
in_addr *dst, struct in_addr *mask)
```

Parameters

→ `rule_h`

Handle to an IPv4 rule. The rule should have been created by the `dagixp_filter_create_ipv4_rule` function.

→ `dst`

Pointer to an `in_addr` structure that contains the destination address to use as the comparand of the filter field.

→ `mask`

Pointer to an `in_addr` structure that contains the destination address mask to use for the filter field.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_get_ipv4_dest_field Function

Purpose Gets the IPv4 destination address used by the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_ipv4_dest_field (RuleH rule_h, struct
in_addr *dst, struct in_addr *mask)
```

Parameters

→ `rule_h`

Handle to an IPv4 rule. The rule should have been created by the `dagixp_filter_create_ipv4_rule` function.

← `dst`

Pointer to an `in_addr` structure that receives the comparand part of the destination address field.

← `mask`

Pointer to an `in_addr` structure that receives the mask part of the destination address field.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_set_ipv6_source_field Function

Purpose Sets the IPv6 source address to filter on.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_set_ipv6_source_field (RuleH rule_h, struct
in6_addr *src, struct in6_addr *mask)
```

Parameters → `rule_h`

Handle to an IPv6 rule. The rule should have been created by the `dagixp_filter_create_ipv6_rule` function.

→ `src`

Pointer to an `in6_addr` structure that contains the source address to use as the comparand of the filter field.

→ `mask`

Pointer to an `in6_addr` structure that contains the source address mask to use for the filter field.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_get_ipv6_source_field Function

Purpose Gets the IPv6 source address used by the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_ipv6_source_field (RuleH rule_h, struct
in6_addr *src, struct in6_addr *mask)
```

Parameters → `rule_h`

Handle to an IPv6 rule. The rule should have been created by the `dagixp_filter_create_ipv6_rule` function.

← `src`

Pointer to an `in6_addr` structure that receives the comparand part of the source address field.

← `mask`

Pointer to an `in6_addr` structure that receives the mask part of the source address field.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_set_ipv6_dest_field Function

Purpose Sets the IPv6 destination address to filter on.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_set_ipv6_dest_field (RuleH rule_h, struct
in6_addr *dst, struct in6_addr *mask)
```

Parameters

→ `rule_h`

Handle to an IPv6 rule. The rule should have been created by the `dagixp_filter_create_ipv6_rule` function.

→ `dst`

Pointer to an `in6_addr` structure that contains the destination address to use as the comparand of the filter field.

→ `mask`

Pointer to an `in6_addr` structure that contains the destination address mask to use for the filter field.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_get_ipv6_dest_field Function

Purpose Gets the IPv6 destination address used by the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_ipv6_dest_field (RuleH rule_h, struct
in6_addr *dst, struct in6_addr *mask)
```

Parameters

→ `rule_h`

Handle to an IPv6 rule. The rule should have been created by the `dagixp_filter_create_ipv6_rule` function.

← `dst`

Pointer to an `in6_addr` structure that receives the comparand part of the destination address field.

← `mask`

Pointer to an `in6_addr` structure that receives the mask part of the destination address field.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_set_ipv6_flow_label_field Function

Purpose Sets the IPv6 flow label to filter on for a given rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_set_ipv6_flow_label_field (RuleH rule_h,
uint32_t flow, uint32_t mask)
```

Parameters → `rule_h`

Handle to an IPv6 rule. The rule should have been created by the `dagixp_filter_create_ipv6_rule` function.

→ `flow`

The value to use as the comparand of the flow label filter field. Only the lower 20 bits of this value is used, the upper 12 bits are ignored.

→ `mask`

The value to use as the mask of the flow label filter field. Only the lower 20 bits of this value is used, the upper 12 bits are ignored.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_get_ipv6_flow_label_field Function

Purpose Gets the IPv6 flow label filter field used by the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_ipv6_flow_label_field (RuleH rule_h,
uint32_t *flow, uint32_t *mask)
```

Parameters → `rule_h`

Handle to an IPv6 rule. The rule should have been created by the `dagixp_filter_create_ipv6_rule function`.

← `flow`

Pointer to a 32-bit value that receives the comparand of the flow label filter field. Only the lower 20 bits of the returned value are valid the upper 12 bits will always be zero.

← `mask`

Pointer to a 32-bit value that receives the mask of the flow label filter field. Only the lower 20 bits of the returned value are valid the upper 12 bits will always be zero.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_set_protocol_field Function

Purpose Sets the IP protocol field to filter on for a given rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_set_protocol_field (RuleH rule_h, uint8_t protocol)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `protocol`

The 8-bit IP protocol number to filter on. Setting this value to either TCP(6), UDP(17), SCTP(132) or ICMP(1) protocol numbers, will allow addition port or ICMP type filter fields to be added to the rule, see to the comments below for more information.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments If the protocol value is set to a TCP(6), UDP(17) or SCTP(132) protocol number, then additional source and destination port filters can be added to the rule, this allows for additional layer 3 level filtering. See the sections on page 24 - 27 for more information on adding port filters to the rule.

If the protocol value is set to the ICMP (1) protocol number, additional ICMP type filters can be added to the rule. See the sections on pages 28 - 30 for more information on adding ICMP type filters to the rule.

There is no mask value associated with the protocol filter field, therefore a direct protocol match must occur for a rule hit.

dagixp_filter_get_protocol_field Function

Purpose Gets the IP protocol filter field used by the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_protocol_field (RuleH rule_h, uint8_t *protocol)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

← `protocol`

Pointer to an 8-bit value that receives the protocol filter field.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_add_source_port_bitmask Function

Purpose Adds a source port bit-masked filter field to the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_add_source_port_bitmask (RuleH rule_h, uint16_t
src, uint16_t mask)
```

Parameters

→ `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `src`

The 16-bit value to use as the comparand of the source port filter field.

→ `mask`

The 16-bit value to use as the mask of the source port filter field.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments This function adds a new bit-masked source port filter to the list of source port filters associated with the rule. Up to 254 source port filter entries (of both bit-masked and range types) are allowed per rule.

The list of source port filters is only loaded into the card and activated (by using the `dagixp_filter_download_ruleset` and `dagixp_filter_activate_ruleset functions`) when the rule's protocol field is set to either TCP, UDP or SCTP values.

Although it is possible to add multiple bit-masked filters to a rule, it is generally not recommended, because when the ruleset is loaded into the card only one bit-masked port filter is allowed per rule. The API compensates for this limitation by duplicating rules that have multiple bit-masked port filters and then allocating one bit-masked port filter per duplicated rule. The situation is worsened if both source and destination port filter lists have bit-masked filter entries, because duplicate rules are generated for every possible combination of source and destination filters.

The more rules that are loaded into the card the worse the filter performance. Generally rules should have either a single bit-masked port filter entry or up to 254 port range filters.

dagixp_filter_add_dest_port_bitmask Function

Purpose Adds a destination port bit-masked filter field to the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_add_dest_port_bitmask (RuleH rule_h, uint16_t
dst, uint16_t mask)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `dst`

The 16-bit value to use as the comparand of the destination port filter field.

→ `mask`

The 16-bit value to use as the mask of the destination port filter field.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments This function is equivalent to `dagixp_filter_add_sorce_port_bitmask` except it adds a destination rather than a source port filter entry. See the comments in the dagixp_filter_add_source_port_bitmask Function section on page 24 for more information.

dagixp_filter_add_source_port_range Function

Purpose Adds a source port range filter field to the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_add_source_port_range (RuleH rule_h, uint16_t
min_port, uint16_t max_port)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `min_port`

The 16-bit value to use as the minimum port value of the source port range.

→ `max_port`

The 16-bit value to use as the maximum port value of the source port range.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments The port range bounded by the `min_port` and `max_port` parameters is inclusive, therefore if you want to filter on a single port value rather than a range, set both the `max_port` and `min_port` parameters to the same value.

Up to 254 source and destination port range filters can be added per rule. Port range filters are not limited to one per rule when loaded into the DAG card like bit-masked port filters are, therefore a number of port range filters can be added to a single rule with a minimal reduction in overall filtering performance.

Bit-masked and range port filters can be mixed within a single rule, however as commented in the dagixp_filter_add_source_port_bitmask Function section on page 24, this is generally not a good idea.

dagixp_filter_add_dest_port_range Function

Purpose Adds a destination port range filter field to the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_add_dest_port_range (RuleH rule_h, uint16_t
min_port, uint16_t max_port)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `min_port`

The 16-bit value to use as the minimum port value of the source port range.

→ `max_port`

The 16-bit value to use as the maximum port value of the source port range.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments This function is equivalent to `dagixp_filter_add_source_port_range` except it adds a destination rather than a source port filter entry. See the comments in the dagixp_filter_add_source_port_range Function section on page25 for more information.

dagixp_filter_get_port_list_count Function

Purpose Returns the number of port filter fields that have been added to the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_port_list_count (RuleH rule_h)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

Returns The return value is a positive number, indicating how many port filters have been added to the rule, if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error code`S.`

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments This function returns the total number of port filters added to the rule, including both source & destination, bit-masked and range type filters. The returned value is typically used to iterate over all the port filters per rule.

dagixp_filter_get_port_list_entry Function

Purpose Returns the details of a port filter at a given index within a rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_port_list_entry (RuleH rule_h, uint32_t
index, port_entry_t * entry)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `index`

The index of the port rule to retrieve.

← `entry`

Pointer to a `port_entry_t` structure that is populated by this function. See the port_entry_t Structure section page 31 for more information.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments This function retrieves the details of a port filter that has been added to a rule, the entry parameter should contain a pointer to a `port_entry_t` structure that will be populated with the details of the rule.

Details for source & destination as well as bit-masked and ranged port filters can be retrieved using this function.

dagixp_filter_add_icmp_type_bitmask Function

Purpose Adds an ICMP type bit-masked filter field to the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_add_icmp_type_bitmask (RuleH rule_h, uint8_t
type, uint8_t mask)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `type`

The 8-bit value to use as the comparand for the ICMP type filter field.

→ `mask`

The 8-bit value to use as the mask of the ICMP type filter field.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments This function adds a new bit-masked ICMP type filter to the list of ICMP type filters associated with the rule. Up to 254 ICMP type filter entries (of both bit-masked and range types) are allowed per rule.

ICMP type filters, target the type field of ICMP headers. IPv6-ICMP (protocol number 58) type packets are not supported by these filters.

The list of ICMP type filters is only loaded into the card and activated (by using the `dagixp_filter_download_ruleset` and `dagixp_filter_activate_ruleset functions`) when the rule's protocol field is set to the ICMP protocol number.

As with port filter lists, it is not recommended to add multiple bit-masked ICMP type rules to a single rule. See the dagixp_filter_add_dest_port_bitmask Function section on page 25 for more information.

dagixp_filter_add_icmp_type_range Function

Purpose Adds an ICMP type range filter field to the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_add_icmp_type_range (RuleH rule_h, uint8_t
min_type, uint8_t max_type)
```

Parameters

→ `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `min_type`

The 8-bit value to use as the minimum ICMP type value of the range.

→ `max_type`

The 8-bit value to use as the maximum ICMP type value of the range.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error cod`es.`

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments The ICMP type range bounded by the `min_type` and `max_type` parameters is inclusive, therefore if you want to filter on a single ICMP type value rather than a range, set both the `max_type` and `min_type` parameters to the same value.

Up to 254 ICMP type range filters can be added per rule. ICMP type range filters are not limited to one per rule when loaded into the DAG card like bit-masked ICMP type filters are, therefore a number of ICMP type range filters can be added to a single rule with a minimal reduction in overall filtering performance.

Bit-masked and range ICMP type filters can be mixed within a single rule, however as commented in the dagixp_filter_add_source_port_bitmask Function section on page 24, this is generally not a good idea.

dagixp_filter_get_icmp_type_list_count Function

Purpose Returns the number of ICMP type filter entries that have been added to the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_icmp_type_list_count (RuleH rule_h)
```

Parameters

→ `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

Returns The return value is a positive number, indicating how many ICMP type filters have been added to the rule, if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_get_icmp_type_list_entry Function

Purpose Returns the details of a ICMP type filter entry at a given index within a rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_icmp_type_list_entry (RuleH rule_h, uint32_t
index, icmp_entry_t * entry)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `index`

The index of the port rule to retrieve.

← `entry`

Pointer to a `icmp_entry_t` structure that is populated by this function. See the icmp_entry_t Structure section on page 31 for more information.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments This function retrieves the details of a ICMP type filter that has been added to a rule, the entry parameter should contain a pointer to a `icmp_entry_t` structure that will be populated with the details of the rule.

port_entry_t Structure

Purpose Used to retrieve the details of a port filter entry within a rule.

Declared In `dagixp_filter.h`

Prototype
```
typedef struct {
	uint32_t	port_type;
	uint32_t	rule_type;
	union {
		struct {
			uint16_t	min_port;
			uint16_t	max_port;
		} bitmask;
		struct {
			uint16_t	value;
			uint16_t	mask;
		} range;
	} data;
} port_entry_t;
```


Field Descriptions

`port_type` : The port type of the filter, this can have one of two possible values; `kSourcePort` or `kDestinationPort.`

`rule_type` : the rule type of the filter, this can have one of two possible values; `kBitmask` or `kRange`.

`min_port` : the minimum value of a port range filter, only valid if the `rule_type` data field is `kRange`.

`max_port` : the maximum value of a port range filter, only valid if the `rule_type` data field is `kRange`.

`Value` : the comparand value of a port bit-mask filter, only valid if the `rule_type` data field is `kBitmask`.

`Mask` : the mask value of a port bit-mask filter, only valid if the `rule_type` data field is `kBitmask`.

icmp_entry_t Structure

Purpose Used to retrieve the details of a ICMP type filter entry within a rule.

Declared In `dagixp_filter.h`

Prototype

```
typedef struct {
       uint32_t     rule_type;
       union {
              struct {
                     uint8_t      min_type;
                     uint8_t      max_type;
              } bitmask;
              struct {
                     uint8_t      value;
                     uint8_t      mask;
              } range;
       } data;
} icmp_entry_t;
```

Field Descriptions

`rule_type` : the rule type of the filter, this can have one of two possible values; `kBitmask` or `kRange`.

`min_type` : the minimum value of a ICMP type range filter, only valid if the `rule_type` data field is `kRange`.

`max_type` : the maximum value of a ICMP type range filter, only valid if the `rule_type` data field is `kRange`.

`Value` : the comparand value of a ICMP type bit-mask filter, only valid if the `rule_type` data field is `kBitmask`.

`Mask` : the mask value of a ICMP type bit-mask filter, only valid if the `rule_type` data field is `kBitmask`.

Filter Rule Attributes

dagixp_filter_set_rule_tag Function

Purpose Sets the tag value associated with the rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_set_rule_tag (RuleH rule_h, uint16_t rule_tag)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `rule_tag`

The new tag value to assign to the rule, only the lower 14 bits of this value are used, the upper 2 bits are always ignored.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments The rule tag is a user defined 14-bit value, which is copied into the color field of the ERF packet record header if the rule produces a hit. Multiple rules can have the same tag value, there is no limitation on the tag value except it must fit within 14-bits.

Although it is allowed, users should avoid setting a rule tag to 0x3FFF, as that is the value defined for the color of packets processed when in loopback mode.

dagixp_filter_get_rule_tag Function

Purpose Retrieves the tag value associated with a rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_rule_tag (RuleH rule_h, uint16_t *rule_tag)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

← `rule_tag`

Pointer to a variable that receives the 14-bit rule tag associated with the rule.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_set_rule_priority Function

Purpose Sets the priority value associated with a rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_set_rule_priority (RuleH rule_h, uint16_t
priority)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `priority`

The 16-bit value to use as the priority for the rule, the lower the priority value the higher the priority of the rule.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments This function sets the priority of a rule within a ruleset, the lower the priority parameter value the higher the rule priority. Rules that have higher priority are compared with incoming packets first.

Care should be taken when iterating over the rules in a ruleset whilst changing the priority of the rule, because rules are priority ordered within a ruleset, changing the priority of a rule will move it's position within a ruleset.

dagixp_filter_get_rule_priority Function

Purpose Retrieves the priority value associated with a rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_rule_priority (RuleH rule_h, uint16_t
*priority)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

← `priority`

Pointer to a variable that receives the 16-bit priority associated with the rule.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_set_rule_action Function

Purpose Sets the action associated with a rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_set_rule_action (RuleH rule_h, action_t action)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `action`

The action to assign to the rule, this can be one of the following constants.

`kReject`: If the rule hits the packet is rejected (dropped).

`kAccept`: If the rule hits the packet is accepted, and routed to either the host or the line based on the rule steering attribute and card configuration.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments The action of the rule determines the course of action to take if the rule produces a hit for a given packet.

dagixp_filter_get_rule_action Function

Purpose Retrieves the action associated with a rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_rule_action (RuleH rule_h, action_t *action)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

← `action`

Pointer to an `action_t` variable that receives the action associated with the rule.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_set_rule_snap_length Function

Purpose Sets the snap length value associated with a rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_set_rule_snap_length (RuleH rule_h, uint16_t
snap_len)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `snap_len`

A 16-bit value that contains the number of bytes to snap the packet length to.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments This function defines the snap length applied to the packet if the rule hits and the rule action is set to kAccept. The snap length only applies if the snap_len value is less than the size of the packet record that produced the rule hit.

The snap length should be a multiple of 8 (this ensures 64-bit alignment of packet records), if not then the snap length is rounded down to the nearest multiple. The minimum snap length allowed is 24 bytes.

The snap length defined for a rule is independent of the snap length specified when configuring the DAG card for packet reception. The snap length that can be configured via the DAG Configuration and Status API (or the `dagconfig` program) is applied to the packet records prior to being present to the IXP filtering software.

dagixp_filter_get_rule_snap_length Function

Purpose Retrieves the snap length associated with a rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_rule_snap_length (RuleH rule_h, uint16_t
*snap_len)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

← `snap_len`

Pointer to a 16-bit variable that receives the snap length associated with the rule.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

dagixp_filter_set_rule_steering Function

Purpose Sets the steering attribute associated with a rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_set_rule_steering (RuleH rule_h, steering_t steering)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

→ `steering`

The steering to assign to the rule, this can be one of the following constants.

`kHost:` The packet record is routed to the host if the rule hits and action attribute is set `kAccept`.

`kLine:` The packet record is routed back out the line if the rule hits and action attribute is set `kAccept.`

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Comments This function defines the target of a packet record if the rule hits and the rule action is set to `kAccept`.

For this option to work successfully the settings for packet routing within the DAG 7.1S card should be set to `kSteerDirectionBit`. See the IXP Filtering API Typical Usage section on page 9 for more information. If the packet routing configuration is set to route packet from the IXP to the host (`kSteerHost`) or line (`kSteerLine`) directly then the steering rule attribute is effectively ignored.

dagixp_filter_get_rule_steering Function

Purpose Retrieves the steering mode associated with a rule.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_get_rule_steering (RuleH rule_h, steering_t *steering)
```

Parameters → `rule_h`

Handle to either an IPv4 or IPv6 rule. The rule should have been created by either the `dagixp_filter_create_ipv4_rule` or `dagixp_filter_create_ipv6_rule` functions.

← `steering`

Pointer to a `steering_t` variable that receives the steering mode associated with the rule.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

Ruleset Loading

The four functions contained within this section will fail unless an open EMA (Embedded Messaging API) connection has been established prior to calling the functions. See the IXP Filtering API and the Embedded Messaging API section on page 7 for more information.

dagixp_filter_download_ruleset Function

Purpose Loads a ruleset into a DAG card.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_download_ruleset (int dagfd, uint8_t iface,
RulesetH ruleset_h)
```

Parameters → `dagfd`

DAG file descriptor provided by `dag_open.`

→ `iface`

This parameter should always contain the kAllInterfaces constant.

→ `ruleset_h`

Handle to the ruleset to load into the card.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

EBADF (bad file descriptor, usually caused by a missing EMA connection)

ERESP (corrupt message received from the IXP)

EUNSUPPORTDEV (the `dagfd` parameter refers to a non-DAG 7.1S card)

Comments Once a ruleset is loaded into a card, it remains in the card until either `dagixp_filter_remove_ruleset` or `dagixp_filter_clear_iface_rulesets` is called, at any time in the future it can be activated by calling `dagixp_filter_activate_ruleset`. The amount of memory available for storing the rulesets on the card is limited, so avoid maintaining a number of rulesets on the card, and remove rulesets from the card when they are no longer needed.

Internally the IXP filtering library maintains a list of ruleset that have been loaded into the card, this list is lost when the library is shutdown. Therefore rulesets that have been downloaded be a previous process can't be reactivated by a new process, similarly it is also recommended to clear all the rulesets on the card when starting a new filter session, this will remove all redundant rulesets left over from a previous process.

dagixp_filter_activate_ruleset Function

Purpose Activates a ruleset on a DAG card.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_activate_ruleset (int dagfd, uint8_t iface,
RulesetH ruleset_h)
```

Parameters → `dagfd`

DAG file descriptor provided by `dag_open`.

→ `iface`

This parameter should always contain the `kAllInterfaces` constant.

→ `ruleset_h`

Handle to the ruleset to activate.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

EBADF (bad file descriptor, usually caused by a missing EMA connection)

ERESP (corrupt message received from the IXP)

EUNSUPPORTDEV (the `dagfd` parameter refers to a non-DAG 7.1S card)

Comments To activate a ruleset on the DAG card, it should have previously been loaded into the card by a successful call to `dagixp_filter_download_ruleset.`

When a ruleset is activated, the rules contained within the ruleset are copied into microcode (packet processing) memory space and the packet processing programs are started. From now onwards packets are processed based on the rules contained in the ruleset. The filtering process is now running independent of the host, for example the ruleset can be deleted and the library closed and the filtering will continue.

It is possible to download a ruleset to the DAG card while it is active, however any changes in the ruleset won't be effective until `dagixp_filter_activate_ruleset` is called on the ruleset again. If a different ruleset is currently activated, this function will deactivate the current one and activate the new one.

The process of activating a filter is not an instantaneous one, during the time it takes, the filtering software is put in loop back mode. See the Modes of Operation section on page 5 for more information.

dagixp_filter_remove_ruleset Function

Purpose Removes a ruleset from the card.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_remove_ruleset (int dagfd, uint8_t iface,
RulesetH ruleset_h)
```

Parameters → `dagfd`

DAG file descriptor provided by `dag_open`.

→ `iface`

This parameter should always contain the `kAllInterfaces` constant.

→ `ruleset_h`

Handle to the ruleset to remove.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

EBADF (bad file descriptor, usually caused by a missing EMA connection)

ERESP (corrupt message received from the IXP)

EUNSUPPORTDEV (the `dagfd` parameter refers to a non-DAG 7.1S card)

Comments This function removes a ruleset from the DAG card and frees the memory allocated for it within the card. This function doesn't destroy the ruleset handle maintained by the API; it just removes it from the card.

Any downloaded ruleset can be removed from the card, including the current active one. If the active ruleset is removed the filtering will continue as before, this function only removes the memory associated with the ruleset on the card, not the actual rules currently used by the packet processing microcode.

dagixp_filter_clear_iface_rulesets Function

Purpose Removes all the rulesets from a DAG card and puts it loop back mode.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_clear_iface_rulesets (int dagfd, uint8_t iface)
```

Parameters → `dagfd`

DAG file descriptor provided by `dag_open`.

→ `iface`

This parameter should always contain the `kAllInterfaces` constant.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

EBADF (bad file descriptor, usually caused by a missing EMA connection)

ERESP (corrupt message received from the IXP)

EUNSUPPORTDEV (the `dagfd` parameter refers to a non-DAG 7.1S card)

Comments This function removes all the ruleset from the DAG card and frees the memory allocated within the card then puts it in loopback mode. This function is typically called when the library is first opened to remove any stale rulesets present in the card.

Statistics

The two functions contained within this section will fail unless an open EMA (Embedded Messaging API) connection has been established prior to calling them. See the IXP Filtering API and the Embedded Messaging API section on page 7 for more information.

statistic_t Type

Purpose Contains the filtering statistic identifiers.

Declared In `dagixp_filter.h`

Prototype

```
typedef enum {
      kPacketsRecv                = 0,
      kPacketsAccepted            = 1,
      kPacketsInPlay              = 1000,
      kPacketsDroppedErfType      = 1001,
      kPacketsDroppedHdlc         = 1002,
      kPacketsDroppedErfSize      = 1003,
      kPacketsDroppedNoFree       = 1004,
      kPacketsDroppedRxFull       = 1005,
      kPacketsDroppedTxFull       = 1006,
      kPacketsDroppedIPv6ExtHeader = 1007,
      kPacketsDroppedSeqBuffer    = 1008,
      kPacketsDroppedTooBig       = 1009,
      kPacketsDroppedMplsOverflow  = 1010,
      kMsfErrUnexpectedSOP        = 1011,
      kMsfErrUnexpectedEOP        = 1012,
      kMsfErrUnexpectedBOP        = 1013,
      kMsfErrNullMPacket          = 1014,
      kMsfErrStatusWord           = 1015,
      kAllStatistics              = 65534,
} port_entry_t;
```

Field Descriptions

`kPacketsRecv`

The number of packets that have been received by the IXP, this is a 64-bit number that will roll over to 0 (after 18446744073709551615 packets have been received).

`kPacketsAccepted`

The number of packets that have been accepted and sent back out the IXP chip, this statistic doesn't cover the number of packets that have been accepted but were dropped because of buffer overflows or transmit errors.

`kPacketsInPlay`

Contains the number of packets that are being filtered at that instant, this statistic will contain a value between 0 and 8192. When the number of packets in play reaches 8192 packets will be dropped, if this statistic is regularly up around 4000 you may want to consider reducing the number of filter rules to avoid packets being dropped due to buffer overrun.

`kPacketsDroppedErfType`

Number of packets dropped because the ERF type received from the FPGA was not a PoS type record (type 1 or 5).

`kPacketsDroppedHdlc`

Number of packets dropped because the HDLC header of the PoS frame was not recognised, the filtering software only accepts a small number of standard HDLC headers.

`kPacketsDroppedErfSize`

Number of packets dropped because of a mismatch in the record length field of the ERF header and the length of the actual record.

`kPacketsDroppedNoFree`

Number of packets dropped due to the internal packet buffers reaching their limit, the filtering software can buffer up to 8192 packets internally when this limit is reached packets are dropped.

This statistic will increase if the host machine can't process the packets fast enough, in such cases reverse flow control will be asserted back to the filtering software and packets will be dropped.

`kPacketsDroppedRxFull`

This statistic is reserved for future use and is currently not implemented.

`kPacketsDroppedTxFull`

This statistic is reserved for future use and is currently not implemented.

`kPacketsDroppedIPv6ExtHeader`

Number of IPv6 packets dropped because an unknown extension header was found in the packet, see to the table on page 3 for a list of supported IPv6 extension headers.

`kPacketsDroppedSeqBuffer`

Number of packets dropped because more than 10 MPLS shim headers were found in the packet, the IP filtering software supports a maximum of 10 MPLS shims.

`kPacketsDroppedTooBig`

Indicates the number of packets dropped because the filtering process delayed the packet to long, this is caused by complex filter rules that lead to varied packet latencies within the filtering software.

`kPacketsDroppedMplsOverflow`

Number of packets dropped because they are too large to be processed, the maximum packet size supported is 4096 bytes including the ERF header.

`kMsfErrUnexpectedSOP`

Number of packets dropped because of an MSF bus error.

`kMsfErrUnexpectedEOP`

Number of packets dropped because of an MSF bus error.

`kMsfErrUnexpectedBOP`

Number of packets dropped because of an MSF bus error.

`kMsfErrNullMPacket`

Number of packets dropped because of an MSF bus error.

`kMsfErrStatusWord`

Number of packets dropped because of an MSF bus error.

`kAllStatistics`

Constant that allows for all the statistics to be cleared with a single function call.

Comments The `kPacketsRecv` and `kPacketsAccepted` statistics are 64-bit counters that roll over when they reach their limit, the rest of the statistics are 32-bit counts that don't roll over.

dagixp_filter_hw_reset_filter_stat Function

Purpose Resets a filtering statistic on a DAG card.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_hw_reset_filter_stat (int dagfd, statistic_t stat)
```

Parameters → `dagfd`

DAG file descriptor provided by `dag_open.`

→ `stat`

The identifier of a statistic.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

EBADF (bad file descriptor, usually caused by a missing EMA connection)

ERESP (corrupt message received from the IXP)

EUNSUPPORTDEV (the `dagfd` parameter refers to a non-DAG 7.1S card)

Comments The `kAllStatistics` constant can be past to this function to reset all the statistics in one go, however internally the API resets one statistic at a time and therefore cannot guarantee that the statistics will all be cleared atomically.

dagixp_filter_hw_get_filter_stat Function

Purpose Gets the contents of a filter statistic from a DAG card.

Declared In `dagixp_filter.h`

Prototype
```
int dagixp_filter_hw_get_filter_stat (int dagfd, statistic_t stat, uint32_t *stat_low, uint32_t *stat_high)
```

Parameters → `dagfd`

DAG file descriptor provided by `dag_open.`

→ `stat`

The identifier of a statistic.

← `stat_low`

Pointer to a 32-bit variable that receives the lower 32-bits of the statistic.

← `stat_high`

Pointer to a 32-bit variable that receives the upper 32-bits of the statistic. This parameter can be NULL if getting a 32-bit statistic counter.

Returns The return value is zero if the operation was successful. Otherwise a negative value is returned and `dagixp_filter_get_last_error` will return one of the following error codes.

ENOTOPEN (the library hasn't been initialised, call `dagixp_filter_startup`)

EINVAL (invalid parameter)

EBADF (bad file descriptor, usually caused by a missing EMA connection)

ERESP (corrupt message received from the IXP)

EUNSUPPORTDEV (the `dagfd` parameter refers to a non-DAG 7.1S card)

Utilities

dagixp_filter_set_last_error Function

Purpose Gets the value of the last error code generated by an IXP filtering API function.

Declared In `dagixp_filter.h`

Prototype `int dagixp_filter_get_last_error (void)`

Parameters This function takes no parameters

Returns The returned value is the last generated error code or 0 if no error was generated.

Comments When any of the `dagixp_filter_functions` are called, they internally reset the last error value to `0`; therefore the last error code will not persist across multiple IXP filtering function calls.

Version History

Version	Date	Reason
1	March 2006	Old Version
2	October 2007	New template and revision
3	November 2008	Corrections and added information about IXP reset.