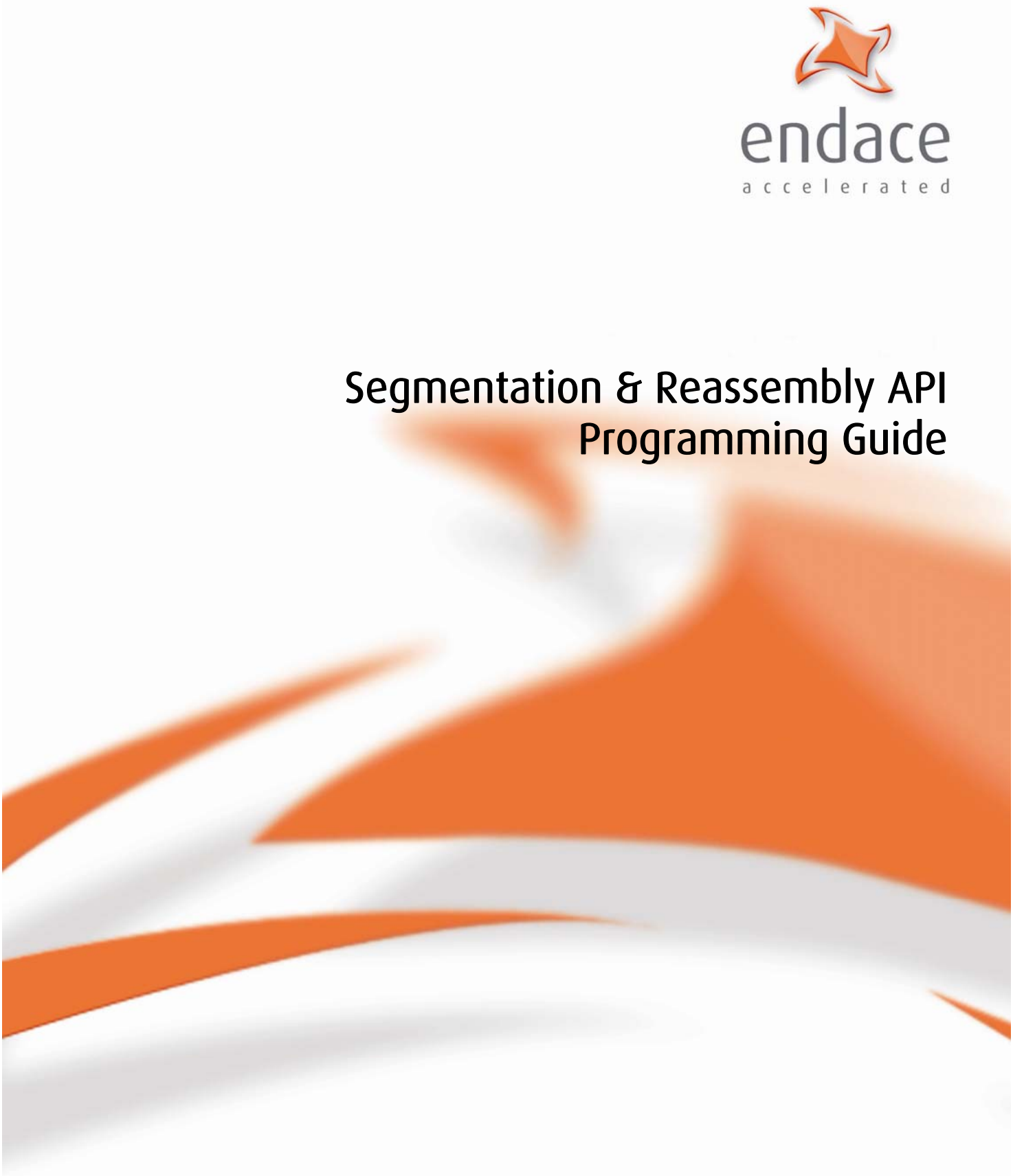


Segmentation & Reassembly API Programming Guide

www.endace.com

Table of Contents

1. Purpose 4
1.1 Revision History 4
2. Introduction 5
2.1 Overview of the AAL reassembler 5
2.2 Notes on DAG3.7T specific functions 5
3. Functional Overview of the AAL Reassembler 6
3.1 Default behavior of the Embedded Software 6
3.1.1 AAL2 specific default behaviour 6
3.1.2 AAL5 specific default behaviour 7
3.2 Activating and Deactivating Virtual Connections 7
3.3 Activating and Deactivating Channels (CID) 7
3.4 Configuration Options 8
3.5 Statistics 8
3.6 Scanning 8
3.7 Filtering 9
3.7.1 General Filtering 9
3.7.2 Extended Filtering (only DAG 3.7T) 9
3.7.3 Extended Filtering Examples 10
4. DAG Sar Function Definitions 12
4.1 dagsar_vci_activate 12
4.1.1 Description 12
4.1.2 Arguments 12
4.1.3 Return Codes 12
4.2 dagsar_vci_deactivate 14
4.2.1 Description 14
4.2.2 Arguments 14
4.2.3 Return Values 14
4.3 dagsar_cid_activate 15
4.3.1 Description 15
4.3.2 Arguments 15
4.3.3 Return Codes 15
4.3.4 DAG3.7T Specific Return Codes 16
4.4 dagsar_cid_deactivate 17
4.4.1 Description 17
4.4.2 Arguments 17
4.4.3 Return Values 17
4.4.4 DAG3.7T Specific Return Codes 17
4.5 dagsar_vci_set_sar_mode 18
4.5.1 Description 18
4.5.2 Arguments 18
4.5.3 Return Codes 18
4.6 dagsar_vci_get_sar_mode 19
4.6.1 Description 19
4.6.2 Arguments 19
4.6.3 Return Codes 19
4.7 dagsar_channel_set_net_mode 20
4.7.1 Description 20
4.7.2 Arguments 20
4.7.3 Return Codes 20
4.8 dagsar_set_buffer_size (only DAG3.7T) 21
4.8.1 Description 21
4.8.2 Return Codes 21
4.9 dagsar_get_stats 22
4.9.1 Description 22
4.9.2 Arguments 22
4.9.3 Return Codes 23
4.10 dagsar_get_interface_stats (deprecated) 24
4.10.1 Description 24
4.10.2 Return Codes 24
4.10.3 DAG3.7T Specific Return Codes 24

4.11 dagsar_reset_stats ... 25
4.11.1 Description ... 25
4.11.2 Arguments ... 25
4.11.3 Return Codes ... 26
4.12 dagsar_reset_stats_all ... 27
4.12.1 Description ... 27
4.12.2 Arguments ... 27
4.12.3 Return Codes ... 27
4.13 dagsar_set_filter_bitmask ... 28
4.13.1 Description ... 28
4.13.2 Arguments ... 28
4.13.3 Examples ... 29
4.13.4 Return Codes ... 29
4.14 dagsar_reset_filter_bitmask ... 30
4.14.1 Description ... 30
4.14.2 Arguments ... 30
4.14.3 Return Codes ... 30
4.15 dagsar_init_scanning_mode ... 31
4.15.1 Description ... 31
4.15.2 Arguments ... 31
4.15.3 Return Codes ... 31
4.16 dagsar_set_scanning_mode ... 32
4.16.1 Description ... 32
4.16.2 Arguments ... 32
4.16.3 Return Codes ... 32
4.17 dagsar_get_scanning_mode ... 33
4.17.1 Description ... 33
4.17.2 Arguments ... 33
4.17.3 Return Codes ... 33
4.18 dagsar_get_scanned_connections_number ... 34
4.18.1 Description ... 34
4.18.2 Arguments ... 34
4.18.3 Return Codes ... 34
4.19 dagsar_get_scanned_connection ... 35
4.19.1 Description ... 35
4.19.2 Arguments ... 35
4.19.3 Return Codes ... 35
5. Data Structures, Attributes, Defines and Enums ... 50
5.1 sar_mode_t ... 50
5.2 net_mode_t ... 50
5.3 stats_t ... 50
5.4 filter_action_t ... 52
5.5 filter_operations_t (only DAG3.7T) ... 52
5.6 dag_filter_level_t (only DAG3.7T) ... 52
5.7 list_operations_t (only DAG3.7T) ... 53
5.8 scanning_mode_t ... **Error! Bookmark not defined.**
5.9 connection_info_t ... **Error! Bookmark not defined.**
6. Data Formats ... 54
6.1 Generic Data Format ... 54
6.2 Multichannel ATM ERF Record ... 56
6.3 Multichannel AAL2 ERF Record ... 57
6.4 Multichannel AAL5 ERF Record ... 58

1. Purpose

This document describes the software interface of the “Segmentation and Reassembly API”. The “Segmentation and Reassembly API” (referred to as SARAPI) runs on the host computer and manages the AAL segmentation and reassembly software running on an embedded processor in a DAG card.

User programs wanting to use the SAR features on a DAG card need other libraries in order to manage the embedded software. The “DAG Embedded Messaging API” (refered to as DAGEMA) it is needed to communicate with the embedded processor. Also the “DAG Configuration API” may be needed to configure the DAG card. This document does not cover other libraries than the SARAPI, although some references will be made to other APIs. Each library has specific documentation, further information can be obtained from them.

Not all DAG cards support segmentation and reassembly of AAL types. this document will explain the support of SAR for the DAG 3.7T and DAG 7.1S cards. There are important architecture differences between these two cards. Differences sometimes are reflected on the software interface. When the behavior is not the same on all the cards, a separate note is added for each of them.

Please be aware that this document is subject to change as additional functionality becomes available. It is recommended to always consult the lastest revision of this document.

1.1 Revision History

Rev.	Date of Change	Description of Change	Revision Originator
0.1	13/March/2006	Initial version. This is a major modification from document “DAG 3.7T Host API”, Cassandra, 20-Dec-2005.	Abel

2. Introduction

2.1 Overview of the AAL reassembler

The AAL reassembler is designed to allow reassembling of AAL5 or AAL2-SSSAR frames on the card without involving the host in processing. The reassembler will receive ATM traffic from the lines, this traffic is then either sent to the host unchanged, dropped or reassembled into AAL5 or AAL2-SSSAR frames, depending on the configuration used. Different configurations can be utilized on different virtual connections, allowing only what data is required to be reassembled, other data can be preserved or rejected.

When reassembly is used, the frame is constructed from the received data. The frame will end when:

1. The reassembly type is AAL5 and the Payload Type Indication (PTI) indicates the end of the frame has been received or, the addition of another ATM cell will cause the buffer allocated to the virtual connection to overflow.

2. The reassembly type is AAL2-SSSAR and the User to User Indication (UUI) indicates that the end of the frame has been received or, the addition of another CPS packet will cause the buffer allocated to the virtual connection and channel (cid) to overflow.

If the frame is AAL5 and has been completed with a PTI end of message indication the length and CRC contained in the trailer is checked. On the DAG 3.7T the AAL5 trailer is removed, on the DAG 7.1S the trailer is kept on the reassembled frame. Errors are indicated in the ERF or Multichannel header.

If the frame is completed due to a buffer overflow it is assumed the length is incorrect. The error is indicated and no attempt is made to check or remove the AAL5 trailer even if the connection is in the AAL5 reassembly mode.

2.2 Notes on DAG3.7T specific functions

Function definitions are described in later chapters of this document. Functions which begin with the prefix *d37t* are available from the DAG3.7T specific library. These functions will be similar, where possible, across different embedded software for the 3.7T, for example, functions for receiving the Software ID will be similar for AAL reassembly, and for IMA.

3. Functional Overview of the AAL Reassembler

3.1 Default behavior of the Embedded Software

After the embedded processor is running, the card will start with all possible Virtual Connections idle. As data is detected on a connection, the virtual connection will be activated, as an ATM cell connection (AAL0), with no reassembly occurring. The virtual connection can be deactivated by using the function **dagsar_vci_deactivate()**. This default behavior can be overridden by changing the SAR mode before data is seen on a connection, this will stop the activation of a ATM Virtual Connection.

All virtual connections default to using the Network to Network Interface (NNI). To use User to Network Interface (UNI), the mode can be changed using the **dagsar_set_net_mode()** function, described later in this document.

DAG 3.7T
Data is returned to the card in bursts. By default the burst size is one mebibyte, this can be changed by using the **aal_msg_set_burst_size()** function. Altering this size can make significant differences to the throughput of the AAL reassembler, depending on the traffic, increasing the size introduces higher latency. This size of the burst should always be larger than the size of all frames to be returned. If the burst size is smaller than the frame size, the frame will not be returned to the host. When the burst size is changed during reassembly, the current block of data will be filled and sent at the previous bust size. All further bursts will be sent at the new size. If the traffic source is to be stopped (for example if the card is to be disconnected) then the **aal_msg_flush_burst_buffer()** function can be used to force any remaining part of the block to be sent to the host. There is also a timeout mechanism when AAL frames are being reassembled. This will cause an incomplete AAL frame to be sent to the burst buffer, if no further cells are seen on the connection for 5 seconds after the previous cell. This is to prevent incomplete AAL frames from being stalled in the card.

DAG 7.1S
Data is returned to the card as soon as it is available. The functions **aal_msg_set_burst_size()** and **aal_msg_flush_burst_buffer()** don't apply to this card. There is no timeout mechanism for reassembled frames. If an AAL2 or AAL5 frame is incomplete, the buffer allocated for that connection will remain used until the frame is completed or the embedded processor is reset.

3.1.1 AAL2 specific default behaviour

DAG 7.1S
still to be defined

When a virtual connection has been set to reassemble AAL2-SSSAR frames the virtual connection can be activated in two different ways. It can be activated by either, **dagsar_vci_activate()** or **dagsar_cid_activate()**. When a virtual connection is activated, in AAL2 reassembly mode, all detected channels(CID) will attempt to reassemble any data that is detected, unless the channel(CID) has been previously deactivated using the **dagsar_cid_deactivate()** function.

The AAL2-SSSAR frame returned will be in an Extensible Record Format(ERF) of TYPE_MC_AAL2. This ERF type is described at the end of this document.

When reassembly is occurring, if the reassembler detects an error in the length of the reassembled frame the error bit will be set in the Multichannel header in the ERF record. The reassembler will then attempt to continue reassembling on the affected channel(CID).

The AAL reassembler will also identify and report MAAL errors as described in the International Telecommunication Unions specification document ITU I.363.2. Currently, the MAAL errors with errnum from 0 to 8 are reported. MAAL error 9 “The CID value in the received CPS-Packet header is not associated with a SAP” is not currently implemented, as any unconfigured CIDs will be reassembled and returned to the host, unless deactivated.

MAAL errors will be reported by sending a MAAL ERF packet. This packet will be of type TYPE_MC_AAL2 (12), and will have the MAAL error indication bit set. The ATM Header field will be completed, and the CID will also be set where appropriate. The first data byte will contain the MAAL errnum, as specified in the specification.

3.1.2 AAL5 specific default behaviour

If a connection has completed reassembling an AAL5 frame, the length of the packet will be checked against the value in the AAL5 trailer. If the length of the frame is incorrect the length error bit will be set, and the frame will be returned without any data removed. If the length is correct the CRC is also checked. On the DAG 3.7T the trailer and padding is removed, on the DAG 7.1S trailer and padding are made available to the user. If the CRC is incorrect this will be indicated in the CRC error indication bit in the Multichannel header, in this case the trailer and padding will still be removed. When working in concatenated modes only the rx_error bit is set on the ERF header flags field, whether it is a length or a CRC error.

The AAL5 frame returned will be in an Extensible Record Format (ERF) of TYPE_AAL5 or TYPE_MC_AAL5 (for concatenated / channelized modes). These ERF types are described at the end of this document.

3.2 Activating and Deactivating Virtual Connections

Activating and deactivating Virtual Connections can be done by using the function **dagsar_vci_activate()** and **dagsar_vci_deactivate().** By default all connections are activated.

When a virtual connection is activated memory will be allocated for the connection. This can be used until the connection is deactivated. It is recommended to deactivate all those connections not being inspected to save memory on the embedded processor. If the embedded processor runs out of memory, attempting to activate more connections will fail.

Any change to a virtual connection, like changing the sar mode or the buffer size (3.7T), first requires the connection to be deactivated. After the change is done the connection can be activated again.

3.3 Activating and Deactivating Channels (CID)

Activating and deactivating Channels to reassemble AAL2-SSSAR frames can be done in a similar way to Virtual Connections, by using the functions **dagsar_cid_activate()** and **dagsar_cid_deactivate().** These functions allow individual CIDs to be turned on and off, within a virtual connection. All cid operations require the Interface, Connection Number, VPI and VCI to be identified, and will only occur on the specified virtual connection.

For these functions to succeed the virtual connection must be put into the AAL2 SAR mode, as described below, prior to an activate or deactivate function call. The individual channels will then be set as requested.

DAG 3.7T
When a CID is activated, memory will be allocated for the connection. This can be used until the connection is deactivated or the virtual connection is deactivated. If the amount of memory allocated needs to be changed, for example, if a connection has been assembling smaller AAL2 frames and will now be assembling AAL2 frames which are larger than the currently allocated buffer, then the CID must be deactivated first. Then the configuration options altered with **dagsar_set_buffer_size()** and finally the connection reactivated. If multiple CIDs are to be changed to a particular size only, one call to **dagsar_set_buffer_size()** is required. Because memory is a limited resource on the DAG3.7T reassembler, it is recommended that only the channels that will be in use should be activated. A total of 128MiB of buffer space is available for Virtual Connections and CIDs attempting to activate more channels when all memory space is used will fail.

3.4 Configuration Options

The AAL reassembler allows some configuration options to be altered.

dagsar_vci_set_sar_mode() allows a virtual connection to be altered between AAL2 and AAL5 reassembly or AAL0 (unchanged).

dagsar_vci_get_sar_mode() allows the current reassembly mode of a Virtual Connection to be retrieved from the card.

dagsar_channel_set_net_mode() sets the UNI or NNI setting for the specified channel.

DAG 3.7T
dagsar_set_buffer_size() allows the amount of memory allocated for new Virtual Connections or channels(CIDs) to be set. The buffer size should be provided in bytes and will only affect new AAL5 or AAL2 CID connections, as they are activated. When **dagsar_set_buffer_size()** is changed, the value is retained until the XScale is reset, or another call to **dagsar_set_buffer_size()** overwrites the value. Note that the Extensible record format types that are used to return the reassembled frames use a 16 bit field to specify the length of the Erf type. This means that the combined length of the Erf header, Multichannel header, ATM header, AAL frame, AAL padding, AAL trailer (when necessary) and 64 bit alignment padding must be equal or less than 64kilebytes in length.

3.5 Statistics

The AAL reassembler provides statistic counters. Values are available via the **dagsar_get_stats()** function, by giving different statistic requested values.

The values will be reset via the **dagsar_reset_stats_all()** function, and by the **dagsar_reset_stats()** function, when the particular statistic is specified. The **dagsar_reset_stats_all()** function should be called at the beginning of a user program to reset all counters. For further information on which statistics are currently available consult the function definitions.

3.6 Scanning

Scanning is used to be automatically aware of the connections present on the physical links. When the scanning mode is set to on, the host software will gather all the observed connections which are currently unconfigured or deactivated and receiving data. Virtual Connections which are currently configured to return data to the host will not be recorded.

The process of scanning begins with initialisation. This allows the system to prepare for recording data and also removes any previous scanning information which may not be correct at the later time. The scanning mode can then be set which will mean the software starts recording the observed connections. During the scanning mode recorded data is not available until scanning mode is turned off again.

During scanning there is an extra load on the system. Although data on configured Virtual Connections may be returned to the host, the AAL reassembler will not necessarily perform correctly or efficiently during scanning. It is therefore not recommended to leave the card in this state during normal operation.

After a period of time, in scanning mode, with data available on some unconfigured or deactivated lines, the scanning mode can then be set to off. When the scanning mode is off the recorded data becomes available. To access the data, first find the total number of entries recorded by calling **dagsar_get_scanned_connections_number().** The entries can then be indexed from 0-(Total number of entries-1). Entries can be received using the **dagsar_get_scanned_connection()** function. All entries will persist until the application is terminated, or the scanning system is initialised again.

3.7 Filtering

All filtering is performed on ATM cells, prior to any other processing. Filtering is not available on reassembled AAL2 or AAL5 frames.

3.7.1 General Filtering

As well as the activation/deactivation of Virtual Connections, there is another possibility to filter cells from the AAL reassembler. Bit masks can be applied to the ATM header and depending if there is a match, the cell can then be discarded. The user has to provide two values to set the filter: a *bit mask* and a *match* value. To each ATM header (4 bytes, HEC is removed) the system calculates the logical AND with the *bit mask*. The result is compared with the *match* value. If they are equal, the result of the match is decided by the value of the *action* argument, which can take two values: *sar_accept* or *sar_reject*. If the values do not match, then the opposite result is taken.

DAG 7.1S
The general filtering, also known as bitmask filtering, is highly recommended to be used. This first line filtering removes extra processing on the embedded processor, making the reassembly process faster and more reliable against packet losses. The bitmask filtering can operate with no problem at 4xOC12 speeds. There is one bitmask filter per interface port, four in total.

3.7.2 Extended Filtering (only DAG 3.7T)

The extended filter module is able to perform simple comparison operations on any contiguous four bytes of data in the ATM cell.

There are two types of filters available. The first is an operational filter which applies a mask to 4 bytes of data, and compares the result to an expected value using a defined filter comparison operator. This is similar to the general SAR filtering described above with the extensions of being able to use other logical operators, and being able to apply the filter operator

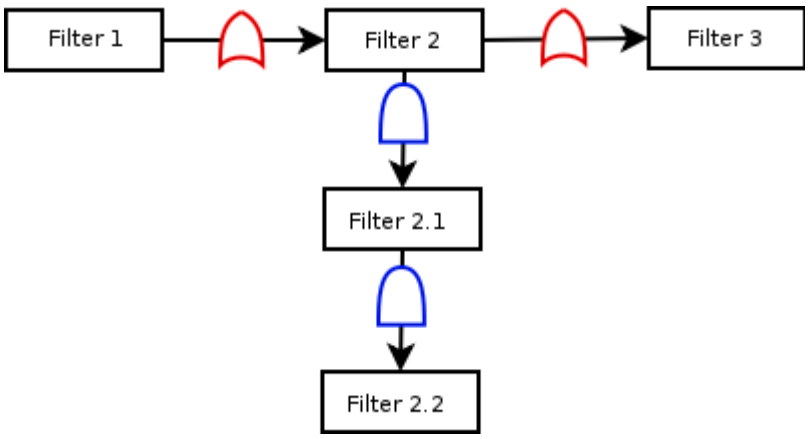
The logical operator is determined after the data and *bit mask* values have been bitwise ANDed together. In general filtering, this is a simple equal operator, where the result must be equal to the *match* value for the filter to be true. In extended filtering, this can also be: not equal, greater than, less than etc. For a full list of the available operators refer to the function definition.

The second type is a history filter. With this filter type, the first comparison occurs between the value given in the *value* argument and the data at *offset* after appling the *bit mask* to the 4 bytes of data at *offset*(using an AND operator). In subsequent comparisons the data at *offset is bitmasked and* compared using the comparison operator to the masked value from the last time the filter run. If the result is a match, the filter is true and *action* is taken. If they do not match then the opposite to *action* is taken.

It is possible to have up to 32 board level filters and 32 channel level filters on each channel configured at any one time. It is not recommended to have more than 256 individual filters set at any time. This recommended limit includes the one possible general filter entry.

3.7.3 Extended Filtering Examples

Extended Filtering also allows multiple filters to be linked together in a tree structure. This allows multiple levels of filters to be constructed by adding sub filters to filters. If for example, a filter structure is created with filters such as this:



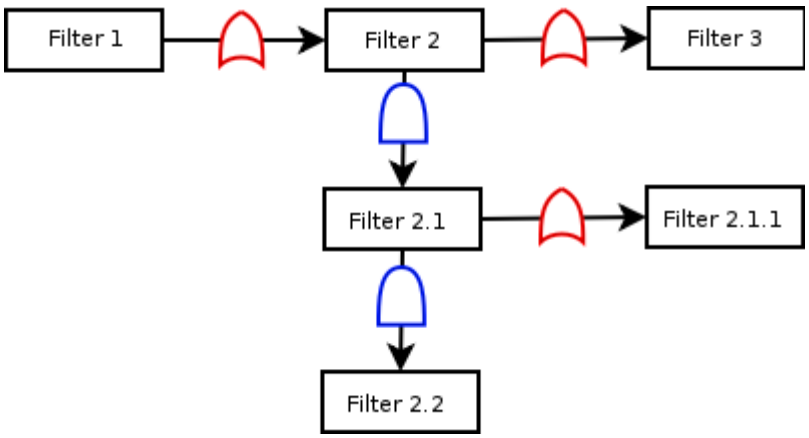
In the above example,

If filter 1 is true, then *action* is taken.

If filter 1 and 2 are false, there is no need to evaluate sub filters 2.1 and 2.2 so filter 3 is evaluated. If it is true *action* is taken, otherwise the opposite of *action* is taken.

If filter 1 is false, and filter 2 is true then sub filters 2.1 and 2.2 are evaluated. If they are both true *action* is taken. If either or both of filters 2.1 and 2.2 are false the opposite of *action* is taken.

To take this concept further, sub filters can be added to sub filters, for example:



With this example, in order for sub filter 2.1 to be true, either of sub filters 2.1 or 2.1.1 needs to be true and if both of them are false then the entire filter 2 sub filter tree is false and filter 3 would then be applied.

When adding sub filters the OR (horizontal) and AND (vertical) structure is preserved. So the parent filter used to add a sub filter to, and the logical operator given will determine the new filters position. For example, in the above example, if a sub filter was added with the parent ID of filter 2, and the logical operator OR, it would become part of the chain made up of filters one, two and three.

If a sub filter was added to filter 2 with the logical operator AND, it would become part of the chain made up of filters two, 2.1 and 2.2. If a sub filter was added to filter 2.1.1, with the logical operator AND, it would become part of a new chain, extending vertically below filter 2.1.1.

If a new sub filter needs to be added to the OR list of filters 2.1 and 2.1.1, then either of these filters may be used as the parent with the logical operator OR.

When deleting a filter all filters that a chained to it will also be deleted, but the chain in which the filter originates will be preserved. In the example above, if filter 2 is deleted, then filters 2.1, 2.1.1 and 2.2 will also be deleted. If filter 2.1 was to be deleted instead, then filter 2.1.1 would be deleted, and filter 2.2 would be preserved in the chain with filter 2.

Although this system allows a large amount of flexibility, it should be noted that adding many filters places additional load on the system. Therefore, it is not efficient to add many filters, unless doing so causes the majority of ATM cells to be dropped.

4. DAG Sar Function Definitions

4.1 dagsar_vci_activate

```
uint32_t dagsar_vci_activate
   uint32_t    dagfd
   uint32_t    iface
   uint32_t    channel
   uint32_t    vpi
   uint32_t    vci
```

4.1.1 Description

The **dagsar_vci_activate()** function sets a virtual connection to return data. This will include allocating memory for use in the reassembly of AAL frames.

DAG 3.7T
The amount of memory allocated will be either the buffer size and an ERF overhead as previously set by **dagsar_set_buffer_size()** or 64kB. The total amount of memory available for reassembling is 128MiB. Attempting to open more connections when the total memory is used will not succeed.

DAG 7.1S
The maximum amount of simultaneously activated connections is 8192. Attempting to open more connections will not succeed.

Note that virtual connections are activated by default. The first packet received from that connection will activate it unless it has been filtered or deactivated before.

4.1.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

iface Specifies the interface of the virtual connection to be activated. In the case of the DAG3.7T card, this should always be zero.

channel Specifies the channel of the virtual connection to be activated. This is the same as the connection number given in the Multichannel ERF header. For concatenated network links this should always be zero.

vpi Specifies the virtual path identifier of the virtual connection to be activated.

vci Specifies the virtual channel identifier of the virtual connection to be activated.

4.1.3 Return Codes

Code	Description
0	Function was successful.
!0	Function was unsuccessful

Code	Description
-1	Memory resources not available (only DAG3.7T)
-2	Virtual Connection is currently activated (only DAG3.7T)
-3	Timeout communicating with the XScale.

Code	Description
-4	Configuration could not be altered (only DAG3.7T)
-5	Unsupported sar_mode type (only DAG3.7T)
-6	Message not transmitted
-7	Message not responded to correctly
-8	Connection Number out of range
-9	VPI out of range
-10	VCI out of range
-11	Interface out of range

4.2 dagsar_vci_deactivate

```
uint32_t dagsar_vci_deactivate
      uint32_t    dagfd
      uint32_t    iface
      uint32_t    channel
      uint32_t    vpi
      uint32_t    vci
```

4.2.1 Description

The **dagsar_vci_deactivate()** function sets a virtual connection to not return any data (all ATM cells will be discarded). This function will also deallocate any memory associated with the virtual connection. If this connection has previously been in the AAL2 reassembly mode then all channels(CID) will return to the unconfigured state.

4.2.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

iface Specifies the interface of the virtual connection to be deactivated. In the case of the DAG3.7T card, this should always be zero.

channel Specifies the channel of the virtual connection to be activated. This is the same as the connection number given in the Multichannel ERF header. For concatenated network links this should always be zero.

vpi Specifies the virtual path identifier of the virtual connection to be deactivated.

vci Specifies the virtual channel identifier of the virtual connection to be deactivated.

4.2.3 Return Values

Value	Description
0	Function was successful.
!0	Function was unsuccessful

Value	Description
-1	Memory resources not available (only DAG3.7T)
-2	Virtual Connection is currently deactivated (only DAG3.7T)
-3	Timeout communicating with the XScale.
-4	Configuration could not be altered (only DAG3.7T)
-6	Message not transmitted
-7	Message not responded to correctly
-8	Connection Number out of range
-9	VPI out of range
-10	VCI out of range
-11	Interface out of range

4.3 dagsar_cid_activate

```
uint32_t dagsar_cid_activate
   uint32_t    dagfd
   uint32_t    iface
   uint32_t    channel
   uint32_t    vpi
   uint32_t    vci
   uint32_t    cid
```

4.3.1 Description

The **dagsar_cid_activate()** function sets an individual AAL2 channel identification (CID) to return data. Before this function will succeed the virtual connection that is described by the arguments channel, vpi and vci must be set to reassemble AAL2 frames. An individual CID may not be activated or deactivated for ATM or AAL5 connections. If the virtual connection described by the arguments channel, vpi and vci is not currently active it will be set to active by use of this function. This could result in other, currently unconfigured CIDs to also return data if there is data present on the line and CID and they have not been previously deactivated. Activating a CID will include allocating memory for use in the reassembly of AAL2-SSSAR frames. The amount of memory allocated will be either the buffer size and an ERF overhead as previously set by **dagsar_set_buffer_size()** or 64kB. The total amount of memory available for reassembling is 128MiB. Attempting to open more connections when the total memory is used will not succeed.

4.3.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

iface Specifies the interface of the CID to be activated. In the case of the DAG3.7T card, this should always be zero.

channel Specifies the channel of the CID to be activated. This is the same as the connection number given in the Multichannel ERF header. For concatenated network links this should always be zero.

vpi Specifies the virtual path identifier of the CID to be activated.

vci Specifies the virtual channel identifier of the CID to be activated.

cid Specifies the channel identifier (CID) to be activated.

4.3.3 Return Codes

Code	Description
0	Function was successful.
!0	Function was unsuccessful

4.3.4 DAG3.7T Specific Return Codes

Code	Description
-1	Memory resources not available
-2	CID is currently activated
-3	Timeout communicating with the XScale.
-5	Unsupported sar_mode type (not AAL2)
-6	Message not transmitted
-7	Message not responded to correctly
-8	Connection Number out of range
-9	VPI out of range
-10	VCI out of range
-11	Interface out of range

4.4 dagsar_cid_deactivate

```
uint32_t dagsar_cid_deactivate
      uint32_t    dagfd
      uint32_t    iface
      uint32_t    channel
      uint32_t    vpi
      uint32_t    vci
      uint32_t    cid
```

4.4.1 Description

The **dagsar_cid_deactivate()** function sets an individual AAL2 channel identifier (CID) to not return any data. All data on the CID will be discarded. This will effect only the CID on the virtual connection specified by the arguments channel, vpi and vci. This will not cause any other CIDs or virtual connections to discard data.

4.4.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

iface Specifies the interface of the virtual connection to be deactivated. In the case of the DAG3.7T card, this should always be zero.

channel Specifies the channel of the virtual connection to be deactivated. This is the same as the connection number given in the Multichannel ERF header. For concatenated network links this should always be zero.

vpi Specifies the virtual path identifier of the virtual connection to be deactivated.

vci Specifies the virtual channel identifier of the virtual connection to be deactivated.

cid Specifies the channel identifier (CID) to be deactivated.

4.4.3 Return Values

Value	Description
0	Function was successful.
!0	Function was unsuccessful

4.4.4 DAG3.7T Specific Return Codes

Value	Description
-1	Memory resources not available
-2	Virtual Connection is currently deactivated
-3	Timeout communicating with the XScale.
-5	Unsupported sar_mode type (not AAL2)
-6	Message not transmitted
-7	Message not responded to correctly
-8	Connection Number out of range
-9	VPI out of range
-10	VCI out of range
-11	Interface out of range

4.5 dagsar_vci_set_sar_mode

```
uint32_t dagsar_vci_set_sar_mode
      uint32_t    dagfd
      uint32_t    iface
      uint32_t    channel
      uint32_t    vpi
      uint32_t    vci
      sar_mode_t  sar_mode
```

4.5.1 Description

The **dagsar_vci_set_sar_mode()** function allows the mode of operation to be changed on a virtual connection. The available modes are ATM, AAL2-SSSAR and AAL5 reassembly. To change the mode a virtual connection must be deactivated before the **dagsar_vci_set_sar_mode()** function is called.

4.5.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

iface Specifies the interface of the virtual connection. In the case of the DAG3.7T card, this should always be zero.

channel Specifies the channel of the virtual connection. This is the same as the connection number given in the Multichannel ERF header. For concatenated network links this should always be zero.

vpi Specifies the virtual path identifier of the virtual connection.

vci Specifies the virtual channel identifier of the virtual connection.

sar_mode Specifies the mode the virtual connection should be changed to. Possible values: {sar_aal0, sar_aal2, sar_aal5}

Of the modes **sar_aal0**, **sar_aal2** and **sar_aal5** are currently valid modes. **sar_error** is used to indicate a virtual connection is in an error mode.

4.5.3 Return Codes

Code	Description
0	Function was successful.
!0	Function was unsuccessful

Code	Description
-1	Memory resources not available (only DAG3.7T)
-2	Virtual Connection is currently activated (only DAG3.7T)
-3	Timeout communicating with the XScale.
-4	Configuration could not be altered (only DAG3.7T)
-6	Message not transmitted
-7	Message not responded to correctly
-8	Connection Number out of range
-9	VPI out of range
-10	VCI out of range
-11	Interface out of range

4.6 dagsar_vci_get_sar_mode

```
sar_mode_t dagsar_vci_get_sar_mode
      uint32_t    dagfd
      uint32_t    iface
      uint32_t    channel
      uint32_t    vpi
      uint32_t    vci
```

4.6.1 Description

The **dagsar_vci_get_sar_mode()** function returns the current mode of operation on a virtual connection.

4.6.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

iface Specifies the interface of the virtual connection. In the case of the DAG3.7T card, this should always be zero.

channel Specifies the channel of the virtual connection. This is the same as the connection number given in the Multichannel ERF header. For concatenated network links this should always be zero.

vpi Specifies the virtual path identifier of the virtual connection.

vci Specifies the virtual channel identifier of the virtual connection.

4.6.3 Return Codes

Code	Description
Valid sar_mode	Function was successful. Possible values: {sar_aal0, sar_aal2, sar_aal5}
Invalid sar_mode	Function was unsuccessful

Code	Description
-3	Timeout communicating with the XScale.
sar_aal0	Virtual Connection is configured to return ATM cells as they are received.
sar_aal2	Virtual Connection will return AAL2 – SSSAR frames as they are reassembled.
sar_aal5	Virtual Connection will return AAL5 frames as they are reassembled.
sar_error	The Virtual Connection has returned an error state.
-6	Message not transmitted
-7	Message not responded to correctly
-8	Connection Number out of range
-9	VPI out of range
-10	VCI out of range
-11	Interface out of range

4.7 dagsar_channel_set_net_mode

```
uint32_t dagsar_vci_set_net_mode
      uint32_t    dagfd
      uint32_t    iface
      uint32_t    channel
      net_mode_t  net_mode
```

4.7.1 Description

The **dagsar_vci_set_net_mode()** sets the net mode of a virtual connection to user to network interface(UNI) or network to network interface (NNI).

4.7.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

iface Specifies the interface of the virtual connection. In the case of the DAG3.7T card, this should always be zero.

channel Specifies the channel of the virtual connection. This is the same as the connection number given in the Multichannel ERF header. For concatenated network links this should always be zero.

net_mode Specifies the net mode the virtual connection should be changed to. Possible values: {uni, nni}.

4.7.3 Return Codes

Code	Description
0	Function was successful
!0	Function was unsuccessful

Code	Description
-1	Memory resources not available. (only DAG3.7T)
-3	Timeout communicating with the XScale.
-4	Configuration could not be altered. (only DAG3.7T)
-6	Message not transmitted
-7	Message not responded to correctly
-8	Connection Number out of range
-9	VPI out of range
-10	VCI out of range
-11	Interface out of range

4.8 dagsar_set_buffer_size (only DAG3.7T)

```
sar_mode_t dagsar_set_buffer_size
      uint32_t    dagfd
      uint32_t    size
```

4.8.1 Description

The **dagsar_set_buffer_size()** function sets the largest expected size for AAL frames on connections or channels to be activated. This size does not include the extra size required for the ERF header, ATM header, AAL5 Trailer or Padding, when required. By changing this size, virtual connections that have been activated previously will not be altered to have the new buffer size. The change will only effect virtual connections or channels that are activated after the buffer size change. To update the buffer size of a previously activated virtual connection or channel, it is necessary to deactivate and re-activate the connection or channel after the buffer size has been altered. If a larger AAL frame is received than the size of the buffer, when the buffer is full it will return the first portion of the AAL frame with the length error bit set. The virtual connection or channel will then keep collecting the AAL frame until the current frame is finished, and return the remaining portion, also with the length error bit set.

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

size The size of the largest AAL frame expected on virtual connections or channels to be activated.

4.8.2 Return Codes

Code	Description
0	Function was successful
!0	Function was unsuccessful

Code	Description
-1	Buffer size is larger than allowable(64 kilobytes).
-2	Buffer size is less than one ATM cell.
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

4.9 dagsar_get_stats

```
uint32_t dagsar_get_stats
      uint32_t    dagfd
      stats_t     statistic
```

4.9.1 Description

The **dagsar_get_stats()** function retrieves the internally held value that corresponds to the requested statistic. The possible statistics currently available for the DAG3.7T are defined by the enumeration stats_t.

4.9.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

statistic The statistic which should be returned

The currently available options for the statistic are:

DAG3.7T	
dropped_cells	This is the number of cells not returned to the host, either due to the cells arriving on a Virtual Connection that is deactivated; or arriving on unconfigured Virtual Connections while in the Scanning mode
filtered_cells	This is the number of cells not returned to the host due to a filter that has determined the cell should be rejected.

DAG7.1S	
dag71s_stats_rx_pkt_0	Number of cells received by the embedded processor before any filtering. Each one of the statistic counters belongs to an interface port.
dag71s_stats_rx_pkt_1	
dag71s_stats_rx_pkt_2	
dag71s_stats_rx_pkt_3	
dag71s_stats_filter_drop_0	Number of cells dropped by the bitmask filters applied to the ATM headers. Each one of the statistic counters belongs to an interface port.
dag71s_stats_filter_drop_1	
dag71s_stats_filter_drop_2	
dag71s_stats_filter_drop_3	
dag71s_stats_to_host_0	Number of cells or frames actually sent to the host. The counter does not differentiate between AAL0, AAL2 or AAL5 frames. Each one of the statistic counters belongs to an interface port.
dag71s_stats_to_host_1	
dag71s_stats_to_host_2	
dag71s_stats_to_host_3	
dag71s_stats_hash_collisions	Number of collisions in the hash table. Only for debugging purposes.
dag71s_stats_datapath_resets	Number of datapath resets. Only for debugging purposes.
dag71s_stats_cmds_exec	Number of user commands executed by the embedded processor. Only for debugging purposes.
dag71s_stats_crc_error_0	Number of CRC errors on AAL5 frames. Each one of the statistic counters belongs to an interface port.
dag71s_stats_crc_error_1	

DAG7.1S	
dag71s_stats_crc_error_2	
dag71s_stats_crc_error_3	
dag71s_stats_loss_counter	Number of packet losses reported by the firmware. This drop counter will increase when the embedded processor cannot process all the incoming packets. The number of packet losses is not accurate, but gives an idea of the performance of the embedded processor. If this statistic counter is greater than zero, it is recommended to decrease the input bandwidth to the embedded processor. This can be achieved through bitmask filters applied to the ATM headers.

Note for DAG7.1S statistic counters: Statistic counters are unsigned 32-bit integers. When the counter reaches the maximum value (0xFFFFFFFF) it will continue counting from the minimum value (0x00000000).

4.9.3 Return Codes

Code	Description
Any	Value of the statistic

Code	Description
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

4.10 dagsar_get_interface_stats (deprecated)

```
uint32_t dagsar_get_interface_stats
      uint32_t    dagfd
      uint32_t    iface
      uint32_t    statistic
```

4.10.1 Description

The **dagsar_get_interface_stats()** function, when used on the DAG3.7T is functionally the same as the **dagsar_get_stats()** function, due to the DAG3.7T not having specified interfaces. This function retrieves the internally held value that corresponds to the requested statistic. This is the total number of cells which fit the criteria for the statistic since the last restart or reset. The possible statistics currently available for the DAG3.7T are defined by the enumeration stats_t. This function is not available when using the DAG7.1S and will not be available in future releases of the SAR API.

The arguments to the function are described below:

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

iface Specifies the interface of the virtual connection to be activated. In the case of the DAG3.7T card, this should always be zero.

statistic The statistic which should be returned

The currently available options for the statistic are:

dropped_cells This is the number of cells not returned to the host, either due to the cells arriving on a Virtual Connection that is deactivated; or arriving on unconfigured Virtual Connections while in the Scanning mode.

filtered_cells This is the number of cells not returned to the host due to a filter that has determined the cell should be rejected.

4.10.2 Return Codes

Code	Description
Any	Value of the statistic

4.10.3 DAG3.7T Specific Return Codes

Code	Description
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

4.11 dagsar_reset_stats

```
uint32_t dagsar_reset_stats
      uint32_t    dagfd
      stats_t     statistic
```

4.11.1 Description

The **dagsar_reset_stats()** function allows a single statistic to be reset to zero without affecting any other statistics, which will continue counting from their current position.

4.11.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

statistic The statistic which should be returned

The currently available options for the statistic are:

DAG3.7T	
dropped_cells	This is the number of cells not returned to the host, either due to the cells arriving on a Virtual Connection that is deactivated; or arriving on unconfigured Virtual Connections while in the Scanning mode
filtered_cells	This is the number of cells not returned to the host due to a filter that has determined the cell should be rejected.

DAG7.1S	
dag71s_stats_rx_pkt_0	Number of cells received by the embedded processor before any filtering. Each one of the statistic counters belongs to an interface port.
dag71s_stats_rx_pkt_1	
dag71s_stats_rx_pkt_2	
dag71s_stats_rx_pkt_3	
dag71s_stats_filter_drop_0	Number of cells dropped by the bitmask filters applied to the ATM headers. Each one of the statistic counters belongs to an interface port.
dag71s_stats_filter_drop_1	
dag71s_stats_filter_drop_2	
dag71s_stats_filter_drop_3	
dag71s_stats_to_host_0	Number of cells or frames actually sent to the host. The counter does not differentiate between AAL0, AAL2 or AAL5 frames. Each one of the statistic counters belongs to an interface port.
dag71s_stats_to_host_1	
dag71s_stats_to_host_2	
dag71s_stats_to_host_3	
dag71s_stats_hash_collisions	Number of collisions in the hash table. Only for debugging purposes.
dag71s_stats_datapath_resets	Number of datapath resets. Only for debugging purposes.
dag71s_stats_cmds_exec	Number of user commands executed by the embedded processor. Only for debugging purposes.
dag71s_stats_crc_error_0	Number of CRC errors on AAL5 frames. Each one of the statistic counters belongs to an interface port.
dag71s_stats_crc_error_1	

DAG7.1S	
dag71s_stats_crc_error_2	
dag71s_stats_crc_error_3	
dag71s_stats_loss_counter	Number of packet losses reported by the firmware. This drop counter will increase when the embedded processor cannot process all the incoming packets. The number of packet losses is not accurate, but gives an idea of the performance of the embedded processor. If this statistic counter is greater than zero, it is recommended to decrease the input bandwidth to the embedded processor. This can be achieved through bitmask filters applied to the ATM headers.

Note for DAG7.1S statistic counters: Statistic counters are unsigned 32-bit integers. When the counter reaches the maximum value (0xFFFFFFFF) it will continue counting from the minimum value (0x00000000).

4.11.3 Return Codes

Code	Description
0	Function was successful
!0	Function was unsuccessful

Code	Description
-2	Statistic is not recognised. (only DAG3.7T)
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

4.12 dagsar_reset_stats_all

```
uint32_t dagsar_reset_stats
      uint32_t    dagfd
```

4.12.1 Description

The **dagsar_reset_stats_all()** function will reset all statistics to zero. It is recommended to call this function at the start of a program that will be using the statistics to put all statistics into a known state.

4.12.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

4.12.3 Return Codes

Code	Description
0	Function was successful
!0	Function was unsuccessful

Code	Description
-2	A Statistic was unable to be identified. (only DAG3.7T)
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

4.13 dagsar_set_filter_bitmask

```
uint32_t dagsar_set_filter_bitmask
      uint32_t          dagfd
      uint32_t          iface
      uint32_t          bitmask
      uint32_t          match
      filter_action_t   filter_action
```

4.13.1 Description

The **dagsar_set_filter_bitmask()** function sets the values of the bitmask, match value and action to be taken for a filter on the card. These values define the filter on the card. Any ATM cells received will have the 32 bits of the ATM header logically AND'd with the bitmask value supplied. The result of this calculation is then compared with the match value, if they are identical, the action defined be filter_action is then taken. The possible filter actions are:

sar_accept Any ATM cells which are the same as the match value after processing the bitmask should be accepted. This will involve passing them onto the Virtual Connections list to determine what reassembly action should be taken. Any ATM cells which are not the same as the match value after filter processing will be discarded immediately

sar_reject Any ATM cells which are the same as the match value after processing the bitmask should be rejected regardless of the Virtual Connection status of the ATM cell. Any ATM cells which are not the same as the match value after filter processing will be processed by the reassembler normally.

4.13.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

iface Specifies the interface where the bitmask filter is going to be applied. In the case of the DAG3.7T card, this should always be zero.

bitmask This is the 32 bit value used by the filter in the mask calculation. This involves performing a logical AND between this value and the ATM header (32 bits not including the HEC).

match The value which the calculation will need to match for the action specified by filter_action to occur. This argument is unused for the DAG3.7T card and should be left at the default.

filter_action The action to be taken when an ATM cell matches the filter value. The contrary to this action is then performed on those ATM cells which do not match the filter value after the bit mask has been applied.

4.13.3 Examples

```
Action: ACCEPT
ATM header:  11001011 11001000 11100111 00100010 0xCBC8E722
Bitmask:     11111111 00000000 00001111 00000000 0xFF000F00
-----------------------------------------------------------
Logical AND: 11001011 00000000 00000111 00000000 0xCB000700
Match value: 11110111 00000000 00000111 00000000 0xF7000700
(does not match -> packet rejected)

Action: REJECT
ATM header:  11001011 11001000 11100111 00100010 0xCBC8E722
Bitmask:     11111111 00000000 00001111 00000000 0xFF000F00
-----------------------------------------------------------
Logical AND: 11001011 00000000 00000111 00000000 0xCB000700
Match value: 11110111 00000000 00000111 00000000 0xF7000700
(does not match -> packet accepted)

Action: ACCEPT
ATM header:  11001011 11001000 11100111 00100010 0xCBC8E722
Bitmask:     11111111 00000000 00001111 00000000 0xFF000F00
-----------------------------------------------------------
Logical AND: 11001011 00000000 00000111 00000000 0xCB000700
Match value: 11001011 00000000 00000111 00000000 0xCB000700
(match -> packet accepted)

Action: REJECT
ATM header:  11001011 11001000 11100111 00100010 0xCBC8E722
Bitmask:     11111111 00000000 00001111 00000000 0xFF000F00
-----------------------------------------------------------
Logical AND: 11001011 00000000 00000111 00000000 0xCB000700
Match value: 11001011 00000000 00000111 00000000 0xCB000700
(match -> packet rejected)
```

4.13.4 Return Codes

Code	Description
0	Function was successful
!0	Function was unsuccessful

Code	Description
-1	Filter could not be set (only DAG3.7T)
-2	The filter was unable to be allocated. This can be due to being out of memory or attempting to add more than 32 filters. (only DAG3.7T)
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

4.14 dagsar_reset_filter_bitmask

```
uint32_t dagsar_reset_filter_bitmask
      uint32_t    dagfd
      uint32_t    iface
```

4.14.1 Description

The **dagsar_reset_filter_bitmask()** function resets the filter on the board so that no further cells are rejected based on the previous filter values.

4.14.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

iface Specifies the interface where the filter is going to be reset. In the case of the DAG3.7T card, this should always be zero.

4.14.3 Return Codes

Code	Description
0	Function was successful
!0	Function was unsuccessful

Code	Description
-1	There is no identifiable filter on the card. (only DAG3.7T)
-2	Filter could not be deleted. (only DAG3.7T)
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

4.15 dagsar_init_scanning_mode

```
uint32_t dagsar_init_scanning_mode
      uint32_t    dagfd
```

4.15.1 Description

The **dagsar_init_scanning_mode()** function initialises the internal set of scanned entries, so that scanning can be performed correctly. This function will remove any current entries and set the number of scanned connections to zero.

This function must be called prior to setting the scanning mode to on. Also the scanning mode must be off for the structure to be initialised. On the DAG3.7T an error will result if this function is called while scanning mode is set to on. On the DAG7.1S no action will be performed on that case.

This function should also be called after scanning has been completed and scanned connections have been processed to free internal memory.

4.15.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

4.15.3 Return Codes

Code	Description
0	Function was successful
!0	Function was unsuccessful

Code	Description
-1	Scanning mode is currently on (only DAG3.7T)
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

4.16 dagsar_set_scanning_mode

```
uint32_t dagsar_set_scanning mode
      uint32_t          dagfd
      scanning_mode_t   scan_mode
```

4.16.1 Description

The **dagsar_set_scanning_mode()** function allows the scanning mode to be turned on and off. During the scanning mode the system will gather information on the available connections which have data available to them and are currently unconfigured or deactivated.

Before turning the scanning mode on the **dagsar_init_scanning_mode()** function must be called. When scanning mode is set to on, no other scanning related functions can be called. Scanning places an extra load on the system which can have a detrimental effect on the performance of the AAL reassembler. For this reason it is not recommended to use the scanning mode whilst in normal operation. scan_error is not a valid option for the card to be set to.

After scanning mode has been initialised, turned on and then turned off the scanned connections can be received using the **dagsar_get_scanned_connections_number()** and **dagsar_get_scanned_connection()** functions.

4.16.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

scan_mode The scanning mode to be set.

4.16.3 Return Codes

Code	Description
0	Function was successful
!0	Function was unsuccessful

Code	Description
-1	Scanning operation could not be completed (only DAG3.7T)
-2	Scanning could not be stopped. (only DAG3.7T)
-3	Timeout communicating with the XScale.
-4	Scanning has not been intialised. (only DAG3.7T)
-5	Scanning could not be started. (only DAG3.7T)
-6	Message not transmitted
-7	Message not responded to correctly

4.17 dagsar_get_scanning_mode

```
scanning_mode_t dagsar_get_scanning_mode
      uint32_t    dagfd
```

4.17.1 Description

The **dagsar_get_scanning_mode()** function returns the current status of the scanning mode as set with the **dagsar_set_scanning_mode()** function.

4.17.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

4.17.3 Return Codes

Code	Description
scan_on	Scanning mode is on
scan_off	Scanning mode is off
scan_error	Scanning mode is undefined
Invalid scanning mode	Function was unsuccessful

Code	Description
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

4.18 dagsar_get_scanned_connections_number

```
uint32_t dagsar_get_scanned connections_number
      uint32_t    dagfd
```

4.18.1 Description

The **dagsar_get_scanned_connections()** function returns the number of connections that the host has information about after scanning. Theses connections can then be accessed using the function **dagsar_get_scanned_connection()**.

Scanned connections are only available after scanning has been performed and prior to the **dagsar_init_scanning_mode()** function being called.

4.18.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

4.18.3 Return Codes

Code	Description
Any	The number of scanned connections available

Code	Description
-1	Scanning mode is set to on.

4.19 dagsar_get_scanned_connection

```
uint32_t dagsar_get_scanned_connection
      uint32_t          dagfd
      uint32_t          connection_number
      connection_info_t *pConnection_info
```

4.19.1 Description

The **dagsar_get_scanned_connection()** function allows information about a connection which was identified during scanning to be retrieved. Connection information is indexed from zero to one less than the value returned by the **dagsar_get_scanned_connections()** function. Connection information returned includes the connection number, VCI, VPI and the Interface number. These identifiers can then be used to configure the connection.

Scanned connections are only available after scanning has been performed and prior to the **dagsar_init_scanning_mode()** function being called.

4.19.2 Arguments

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

connection_number The index of the connection information to be retrieved.

pConnection_info Pointer to the structure that will contain the connection information after a successful call to the function. This pointer must already be pointing to a valid connection_info structure, and any memory used must be maintained and deallocated by the user.

4.19.3 Return Codes

Code	Description
0	Function was successful
!0	Function was unsuccessful

Code	Description
-1	Scanning mode is set to on.
-2	connection_number requested is out of range.
-3	Error attempting to access table entry (only DAG3.7T)
-4	Connection info pointer was not initialized. (only DAG7.1S)

5. DAG3.7T Specific Functions

Note: These functions are not available when using the DAG7.1S.

5.1 d37t_write_software_id

```
int d37t_write_software_id
   int         dagfd
   int32_t     num_bytes
   uint8_t     *datap
   uint32_t    key
```

5.1.1 Description

The **d37t_write_software_id()** function writes a new software ID into the EEPROM attached to the DAG3.7T's XScale processor. The key field is a simple security feature to prevent accidental access to the EEPROM; if the key does not match a specific value in the embedded software on the XScale then the XScale will freeze and a restart will be required.

The length of the ID must be at least 1 byte and no more than 128 bytes. (On the Rev B and Rev C DAG3.7T boards the EEPROM can store up to 128 bytes of data).

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

num_bytes The number of bytes to write into the EEPROM. Between 1 and 128 inclusive. If the number is less than 128 then the remaining space is filled with 0.

datap Points to a byte array containing the data to write to the EEPROM.

key The write-enable key. This must be specified to enable write-access to the EEPROM. If the key is incorrect then EEPROM will not be written to and the XScale will lock up.

5.1.2 Return Codes

Code	Description
0	Function was successful.
-1	invalid number of bytes specified
-2	Firmware error writing to the EEPROM.
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

5.2 d37t_read_software_id

```
int d37t_read_software_id
   int          dagfd
   int32_t      num_bytes
   uint8_t      *datap
```

5.2.1 Description

The **d37t_read_software_id()** function reads the software ID from the EEPROM attached to the DAG3.7T's XScale processor. The num_bytes field specifies how many bytes to read from the EEPROM. The contents are stored in the byte array pointed to by datap.

The length of the ID must be at least 1 byte and no more than 128 bytes. (On the Rev B and Rev C DAG3.7T boards the EEPROM can store up to 128 bytes of data).

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

num_bytes The number of bytes to read from the EEPROM. Between 1 and 128 inclusive.

datap Points to a byte array to be used to return the contents of the EEPROM.

5.2.2 Return Codes

Code	Description
0	Function was successful.
-1	invalid number of bytes specified
-2	Firmware error reading to the EEPROM.
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

5.3 d37t_read_version

```
int d37t_read_version
   int         dagfd
   uint32_t    *version
   uint32_t    *type
```

5.3.1 Description

The **d37t_read_version()** function reads the version and type from the program running in the DAG3.7T's XScale processor.

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

version The version number of the program currently running on the scale processor.

type Type of software running on the XScale processor. For the AAL reassembler this value will be equal to the defined variable AAL (numerical value 2).

5.3.2 Return Codes

Code	Description
0	Function was successful.
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

5.4 d37t_read_temperature

```
int d37t_read_temperature
   int         dagfd
   int32_t     sensor_id
   int         *temperature
```

5.4.1 Description

The **d37t_read_temperature()** function reads the current temperature from the LM63 temperature sensor device attached to the DAG3.7T's XScale processor.

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

sensor_id Specifies which temperature sensor to read. Rev B and Rev C DAG3.7T boards only have one so set this to 0. (0=default).

temperature Points to a integer to store the temperature reading in.

5.4.2 Return Codes

Code	Description
0	Function was successful.
-1	Invalid sensor ID.
-2	Firmware error reading the temperature sensor.
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

5.5 aal_msg_set_burst_size

```
int aal_msg_set_burst_size
   uint32_t    dagfd
   uint32_t    size
```

5.5.1 Description

The **aal_msg_set_burst_size()** is a DAG3.7T AAL reassembler specific function that allows the amount of data that is written to the host at a time to be set. The burst size will not be altered in the block which is filling currently. All subsequent blocks will be at the size specified if the function is successful. This size must be larger than the largest possible AAL frame, including headers and padding expected on all connections. If a frame is received that is larger than this size it will not be returned to the host.

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

size The size of the burst to be sent in bytes. This can range between 72 and 10485760 (10MiB) and must be a multiple of 8 bytes.

5.5.2 Return Codes

Code	Description
0	Function was successful.
-2	Size argument was out of range.
-3	Timeout communicating with the XScale.
-4	Size argument was not a multiple of 8 bytes.
-6	Message not transmitted
-7	Message not responded to correctly

5.6 aal_msg_flush_burst_buffer

```
uint32_t aal_msg_flush_burst_buffer
   uint32_t    dagfd
```

5.6.1 Description

The **aal_msg_flush_burst_buffer()** function send any unfinished bursts of data to the host, regardless of the amount of data currently held for the burst. This function can be used if the card is to be disconnected from the traffic source and all data is required at the host. This function should not be used when further data is expected. If the latency of the data is not suitable for an application the recommended solution is to reduce the burst size with the **aal_msg_set_burst_size()** function rather than using this function.

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

5.6.2 Return Codes

Code	Description
0	Function was successful.
-1	No data is available at the Reassembler to send
-2	No memory has been allocated to send
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

5.7 aal_msg_set_filter

```
uint32_t aal_msg_set_filter
   uint32_t    dagfd
   uint32_t    offset
   uint32_t    mask
   uint32_t    value
   uint32_t    operation
   uint32_t    filter_level
   uint32_t    level_conf
   uint32_t    history
   uint32_t    priority
```

5.7.1 Description

The **aal_msg_set_filter()** function sets a filter according to the settings given. For an overview of the filter operation, consult the overview section at the beginning of this document. A maximum of 32 board level filters and 32 channel level filters per channel can be configured at any time for the AAL reassembler, although it is recommended to have a maximum of 256 filters at any given time.

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

offset The offset refers to the start location of the four bytes to be filtered on for this filter. The offset in measured in bytes and starts from the very beginning of the ERF that the ATM cell is contained in. Therefore an offset of zero would filter on the section of the timestamp of the ATM ERF. An offset of 20 corresponds to the ATM header. The offset must specify 4 bytes which will fall inside the ATM ERF, therefore no values greater than 68 will be valid.

mask This is the 32 bit value used by the filter in the mask calculation. This involves performing a logical AND between this mask value and the data located in the ATM ERF at *offset.*

value When an operational filter is used, this is the value that the result of ANDing the data in the ATM ERF at *offset* with the *mask* value is compared to. This will determine whether the filter is true or false. In the case of a history filter, this is the value that the data at *offset* will be compared to during the first comparison. After this, the value that was calculated in the previous application of this filter is used.

operation This the operation used to compare the data at *offset* to the *value* argument. This filter can be equal, not equal, greater than or equal, less than or equal, greater than, less than, AND, OR or XOR. The enumeration filter_operations_t specifies the numerical values that correspond to these operations.

filter_level A filter can be set at the Board level, where the filter will be applied to all data coming in to the board (DAG_FILTER_LEVEL_BOARD) , or the filter can be set at the connection level (DAG_FILTER_LEVEL_CHANNEL), where only data arriving with the specified connection number will be have the filter applied. To specify which connection to filter on use the level_conf argument.

level_conf This the the indication of which connection (0-511) to filter on. This argument is only valid when the filter level is set to DAG_FILTER_LEVEL_CHANNEL during filter initialisation. At all other times this argument should be set to zero.

history This is a boolean value which determines if this filter will be a history or an operational filter. A value of true (1) will cause this to become a history filter.

priority The priority allows a heirachy of filters to be set up. When adding filters, they will be added in the priority order given. Therefore, to add a filter that should only be evaluated, if other filters have already been evaluated, add the filter with a lower priority number.

5.7.2 Return Codes

Code	Description
0	Function was unsuccessful.
!0	The unique filter identifier number. This can be used to reference the filter in future calls to delete the filter.
-2	The filter was unable to be allocated. This can be due to being out of memory or attempting to add more than 32 filters at either board level or channel level.
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

5.8 aal_msg_set_subfilter

```
uint32_t aal_msg_set_subfilter
   uint32_t          dagfd
   uint32_t          offset
   uint32_t          mask
   uint32_t          value
   uint32_t          operation
   uint32_t          filter_level
   uint32_t          level_conf
   uint32_t          history
   uint32_t          priority
   uint32_t          parent_id
   list_operations_t list_operation
```

5.8.1 Description

The **aal_msg_set_subfilter()** function sets a filter according to the settings given, in a sub filter list below the parent filter specified. The sub filter list is created with the option of being an AND or OR list. For an overview of the sub filter operation, consult the overview section at the beginning of this document. A maximum of 32 board level filters and 32 channel level filters per channel can be configured at any time for the AAL reassembler, although it is recommended to have a maximum of 256 filters at any given time.

dagfd The file descriptor for the DAG card as returned from dag_open(). The card should have been initialized via dagema_open_conn().

offset The offset refers to the start location of the four bytes to be filtered at. The offset is measured in bytes, and starts from the very beginning of the ERF that the ATM cell is contained in. Therefore, an offset of zero would start filtering at the ERF timestamp. An offset of 20 corresponds to the start of the ATM header. The offset must specify 4 bytes which will fall inside the ATM ERF, therefore no values greater than 68 will be valid.

mask This is the 32 bit value used by the filter in the mask calculation. This involves performing a logical AND between this mask value and the data located in the ATM ERF at *offset.*

value When an operational filter is used, this is the value that the result of ANDing the data in the ATM ERF at *offset* with the *mask* value is compared to. This will determine whether the filter is true or false. In the case of a history filter, this is the value that the data at *offset* will be compared to during the first comparison. After this, the value that was calculated in the previous application of this filter is used.

operation This the operation used to compare the data at *offset* to the *value* argument. This filter can be equal, not equal, greater than or equal, less than or equal, greater than, less than, AND, OR or XOR. The enumeration filter_operations_t specifies the numerical values that correspond to these operations.

filter_level A filter can be set at the Board level, where the filter will be applied to all data coming in to the board (DAG_FILTER_LEVEL_BOARD), or the filter can be set at the connection level (DAG_FILTER_LEVEL_CHANNEL), where only data arriving with the specified connection number will be have the filter applied. To specify which connection or line to filter on use the level_conf argument.

level_conf This determines which connection (0-511) to filter on. This argument is only valid when the filter level is set to DAG_FILTER_LEVEL_CHANNEL during filter initialisation. At all other times this argument should be set to zero.

history This is a boolean value which determines if this filter will be a history or an operational filter. A value of true (1) will cause this to become a history filter.

priority The priority allows a hierarchy of filters to be set up. When adding filters, they will be added in the priority order given. Therefore, to add a filter that should only be evaluated only when the other filters have already been evaluated, add the filter with a lower priority number then the other filters.

parent_id This is the identifier of the filter that will be used as the starting point of this sub filter. This argument should never be zero.

list_operation The list_operation specifies if the filter to be created should be in an AND or OR list with the parent filter.

5.8.2 Return Codes

Code	Description
0	Function was unsuccessful.
!0	The unique filter identifier number. This can be used to reference the filter in future calls to delete the filter.
-2	The filter was unable to be allocated. This can be due to being out of memory or attempting to add more than 32 filters at board level, or channel level per channel.
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

5.9 aal_msg_set_filter_action

```
int aal_msg_set_filter_action
   uint32_t    dagfd
   uint32_t    action
```

5.9.1 Description

The **aal_msg_set_filter_action()** function sets the action that will be taken on ATM cells which positively match any filters which are in place. Conversely the opposite of the action will be taken on any ATM cells which do not match the filters. This is a global action that occurs on the results of all set filters. By default the reject option is set.

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

action The action to be taken when an ATM cell matches the filter value. The opposite action is performed on those ATM cells which do not match the filter value after the bit mask has been applied.

The possible values for the action of the filter are:

sar_accept (0) Any ATM cells which are the same as the match value after processing the bitmask should be accepted. This will involve passing them onto the Virtual Connections list to determine what reassembly action should be taken. Any ATM cells which are not the same as the match value after filter processing will be discarded immediately

sar_reject (1) Any ATM cells which are the same as the match value after processing the bitmask should be rejected, regardless of the Virtual Connection status of the ATM cell. Any ATM cells which are not the same as the match value after filter processing will be processed by the reassembler normally.

5.9.2 Return Codes

Code	Description
0	Function was successful.
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

5.10 aal_msg_reset_filter

```
uint32_t aal_msg_reset_filter
   uint32_t    dagfd
   uint32_t    filter_id
```

5.10.1 Description

The **aal_msg_reset filter()** function allows a single filter to be deleted from the list of filters, without effecting any other filters. This function should not be used when attempting to delete a filter created with the more general **dagsar_set_filter_bitmask()** function.

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

filter_id The unique identifier of the filter to be deleted as returned by the **aal_msg_set_filter()** function.

5.10.2 Return Codes

Code	Description
0	Function was successful.
-2	Filter could not be deleted.
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

5.11 aal_msg_reset_all_filters

```
int aal_msg_reset_all_filters
   uint32_t    dagfd
```

5.11.1 Description

The **aal_msg_reset_all_filters()** function removes all filters set, including those set with the more general **dagsar_set_filter_bitmask()** function.

dagfd The file descriptor for the DAG card as returned from dag_open(). This card also should have been initialized via dagema_open_conn().

5.11.2 Return Codes

Code	Description
0	Function was successful.
-3	Timeout communicating with the XScale.
-6	Message not transmitted
-7	Message not responded to correctly

7. Data Structures, Attributes, Defines and Enums

7.1 sar_mode_t

The sar_mode_t enumeration is defined in the dagsarapi.h file and is in the form

```
typedef enum {
 sar_aal0,
 sar_aal2,
 sar_aal5,
 sar_error
}sar_mode_t;
```

Of these modes sar_aal0, sar_aal2 and sar_aal5 are currently valid modes. sar_error is used to indicate a virtual connection is in an error mode.

7.2 net_mode_t

The sar_mode_t enumeration is defined in the dagsarapi.h file and is in the form

```
typedef enum
{
    nni,
    uni
}net_mode_t;
```

These modes can be set using the **dagsar_set_net_mode()** function.

7.3 stats_t

The stats_t enumeration is defined in the dagsarapi.h file and is in the form

```
typedef enum
{
 dropped_cells,
 filtered_cells,

   dag71s_stats_rx_pkt_0 = 0x100,
   dag71s_stats_rx_pkt_1,
   dag71s_stats_rx_pkt_2,
   dag71s_stats_rx_pkt_3,
   dag71s_stats_filter_drop_0,
   dag71s_stats_filter_drop_1,
   dag71s_stats_filter_drop_2,
   dag71s_stats_filter_drop_3,
   dag71s_stats_to_host_0,
   dag71s_stats_to_host_1,
   dag71s_stats_to_host_2,
   dag71s_stats_to_host_3,
   dag71s_stats_hash_collisions,
   dag71s_stats_datapath_resets,
   dag71s_stats_cmds_exec,
   dag71s_stats_crc_error_0,
   dag71s_stats_crc_error_1,
   dag71s_stats_crc_error_2,
   dag71s_stats_crc_error_3,
   dag71s_stats_loss_counter
}stats_t;
```

This defines the currently available statistics that can be received from the AAL reassembler. A definition of what these statistics represent can be found in the dagsar function definition section with the information for the function **dagsar_get_stats()**.

7.4 scanning_mode_t

The scanning_mode_t enumeration is defined in the dagsarapi.h file and is in the form

```
typedef enum
{
 scan_on,
 scan_off,
 scan_error
}scanning_mode_t;
```

This defines the scanning modes which can be set and received using the **dagsar_set_scanning_mode()** and **dagsar_get_scanning_mode()** functions. Although scan_error is defined, it should not be used as a mode to be set.

7.5 connection_info_t

The connection_info_t structure is defined in the dagsarapi.h file and is in the form

```
typedef struct
{
 uint32_t iface;
 uint32_t channel;
 uint32_t vpi;
 uint32_t vci;
}connection_info_t;
```

This structure defines the information which will be returned from a call to the **dagsar_get_scanned_connection()** function. On the DAG3.7T card the interface is not a valid identifier. This connection information can then be used to configure or activate a channel. The channel member of the structure is the same as the connection number given in the Multichannel ERF header.

8. DAG3.7T Specific Data Structures, Attributes, Defines and Enums

Note: The following data structures, attruibutes defines and/or Enums are not available when using the 7.1S card.

8.1 filter_action_t

The filter_action_t enumeration is defined in the dagsarapi.h file and is in the form

```
typedef enum
{
 sar_accept,
 sar_reject
}filter_action_t;
```

This defines the action which can be taken with a cell which is identified as fitting the set filter requirements. These actions can be set using the **dagsar_set_filter_bitmask()** function.

8.2 filter_operations_t

The filter_operations_t structure is defined in the aal_config_msg.h file and is in the form

```
typedef enum
{
 DAG_EQ = 0, /* 0 equal */
 DAG_NEQ, /* 1 not equal */
 DAG_LE, /* 2 less than or equal */
 DAG_GE, /* 3 greater than or equal */
 DAG_LT, /* 4 less than */
 DAG_GT, /* 5 greater than */
 DAG_AND, /* 6 bitwise and */
 DAG_OR, /* 7 bitwise or */
 DAG_XOR, /* 8 bitwise exclusive-or */

 DAG_NUM_FILTER_OPERATIONS
}connection_info_t;
```

This defines the actions which can be used in extended filtering to compare the masked data value with the match value. This is only for an operational filter, the History filter always compares the masked data value with the previous masked data value, and if they are equal, then action is taken.

8.3 dag_filter_level_t

The dag_filter_level_t structure is defined in the aal_config_msg.h file and is in the form

```
typedef enum
{
 DAG_FILTER_LEVEL_BOARD = 0,
 DAG_FILTER_LEVEL_CHANNEL,

 DAG_NUM_FILTER_LEVELS /* this has to be the last in list */

}DAG_filter_level_t;
```

This defines the level on which a filter is to be set.

8.4 list_operations_t

The list_operations_t structure is defined in the aal_config_msg.h file and is in the form

```
typedef enum
{
 DAG_OR_LIST = 0,
 DAG_AND_LIST

}list_operations_t;
```

This allows subfilter lists to be added in a form where either all of the filters need to be true to pass (DAG_AND_LIST) or any of the filters need to be true to pass (DAG_OR_LIST).

9. Data Formats

9.1 Generic Data Format

Data received from the AAL reassembler is transmitted from the card in Extensible Record Format (ERF). The Generic ERF Format is as follows:

BYTE 3	BYTE 2	BYTE 1	BYTE 0
timestamp			
timestamp			
type	flags	rlen	
lctr		wlen	
(rlen - 16) bytes of record			

Timestamp	The time of arrival of this cell. Timestamps are in little-endian byte order (Pentium native). All other fields are big-endian byte order. No byte reordering is done on the Payload.
Type	This field contains an enumeration of the frame subtype. Valid types for the AAL reassembler on the DAG3.7T card are: 7: TYPE_MC_ATM 9: TYPE_MC_AAL5 Valid types for the AAL reassembler on the DAG7.1S card are: 3: TYPE_ATM (ATM cells) 4: TYPE_AAL5 (reassembled AAL5 frames) 7: TYPE_MC_ATM (multichannel ATM cells) 9: TYPE_MC_AAL5 (multichannel reassembled AAL5 frames) 12: TYPE_MC_AAL2 (multichannel reassembled AAL2 frames)
Flags	This byte is divided into 2 parts, the interface identifier, and the capture offset. 1-0: capture interface 0-3 2: varying record lengths present 3: truncated record [insufficient buffer space] 4: rx error [link error] 5: 5: ds error [internal error] 7-6: reserved
Rlen: Record Length	Total length of the record transferred over PCI bus to storage.
Lctr: Loss Counter	A 16 bit counter, recording the number of packets lost between the DAG card and the memory hole due to overloading on the PCI bus. The counter starts at zero, and sticks at 0xffff.

Timestamp	The time of arrival of this cell. Timestamps are in little-endian byte order (Pentium native). All other fields are big-endian byte order. No byte reordering is done on the Payload.
Wlen: Wire Length	Packet length including some protocol overhead. The exact interpretation of this quantity depends on the physical medium.

9.2 Multichannel ATM ERF Record

The Multichannel ATM ERF record is a fixed length record in the following format.

BYTE 3	BYTE 2	BYTE 1	BYTE 0
timestamp			
timestamp			
type: 7	flags	rlen	
lctr		wlen	
Multichannel Header			
ATM Header			
48 bytes of record			

All fields are the same as the Generic Record Format unless listed below.

Flags	Capture interface is always zero. RX Error is set if any MC header Error bit is set.
Multichannel Header	This header is divided into several bit fields. Some of these fields are not used by the AAL reassembler but are described here for continuity. 0-9 Connection number (0-1023) (512 connections are supported by DAG3.7T card, 672 for the DAG7.1S) 10-14 Reserved 15 Multiplexed from IMA group into ATM stream 16-19 Physical port (0-15) cell was captured on 20-23 Reserved 24 Lost Byte Error. The internal data path had an unrecoverable error. 25 HEC corrected 26 OAM Cell CRC-10 Error (Not Implemented) 27 OAM Cell 28 1st Cell. This is the first cell received since this connection was configured. 29- 31 Reserved
ATM Header	This does not include the 8-bit HEC

9.3 Multichannel AAL2 ERF Record

The Multichannel AAL2 ERF record is in the following format.

BYTE 3	BYTE 2	BYTE 1	BYTE 0
timestamp			
timestamp			
type: 12	flags	rlen	
lctr		wlen	
Multichannel Header			
ATM Header			
(rlen - 24) bytes of record			

All fields are the same as the Generic Record Format unless listed below.

Flags	Capture interface is always zero. RX Error is set if any MC header Error bit is set.
Wlen: Wire Length	This contains the length of the AAL2 frame including the ATM Header but not including the ERF Header. The ERF record will always be 64 bit aligned, if the AAL2 frame is not 64 bit aligned the record will be padded at the end of the record with the value 0x00. This padding will not be included in the Wlen count.
Multichannel Header	This header is divided into several bit fields. Some of these fields are not used by the AAL reassembler but are described here for continuity. 0-9 Connection number (0-1023) (512 connections are supported by DAG3.7T card) 10-15 Reserved 16-19 Physical port (0-15) cell was captured on 20- 27 Reserved 28 1st Cell. This is the first cell received since this connection was configured. 29- 31 Reserved
ATM Header	This does not include the 8-bit HEC

9.4 Multichannel AAL5 ERF Record

The Multichannel AAL5 ERF record is in the following format.

BYTE 3	BYTE 2	BYTE 1	BYTE 0
timestamp			
timestamp			
type: 9	flags	rlen	
lctr		wlen	
Multichannel Header			
ATM Header			
(rlen - 24) bytes of record			

All fields are the same as the Generic Record Format unless listed below.

Flags	Capture interface is always zero. RX Error is set if any MC header Error bit is set.
Wlen: Wire Length	This contains the length of the AAL5 frame including the ATM Header but not including the ERF Header. The ERF record will always be 64 bit aligned, if the AAL5 frame is not 64 bit aligned the record will be padded at the end of the record with the value 0x00. This padding will not be included in the Wlen count.
Multichannel Header	This header is divided into several bit fields. Some of these fields are not used by the AAL reassembler but are described here for continuity. 0-9 Connection number (0-1023) (512 connections are supported by DAG3.7T card) 10-15 Reserved 16-19 Physical port (0-15) cell was captured on 20 CRC checking available 21 CRC Error 22 Length checking available 23 Length Error 24- 27 Reserved 28 1st Cell. This is the first cell received since this connection was configured. 29- 31 Reserved
ATM Header	This does not include the 8-bit HEC