# DAG 3.7T Card User Guide

EDM01-12

## 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

# Contents

# Introduction

## Overview

The Endace DAG 3.7T Card is optimized for monitoring and interception with precise timestamping capability on up to 16 T1/E1 network links. The DAG card actively manages the movement of network data into memory while only consuming a minimal amount of the host computers resources.

The DAG 3.7T is a 16 port, PCI card that allows capture and transmission of data.

Supported protocols include raw data (unmapped/unframed), HDLC and ATM over as many as 512 sub-channels, channels and hyper-channels. The DAG 3.7T also supports Inverse Multiplex ATM (IMA) link aggregation, AAL2 and AAL5 segmentation and reassembly, and frame (HDLC), cell (ATM) and packet (AAL2/5) filtering based on user-defined filter rules. An onboard Intel® XScale™ processor provides the means to pre-process data prior to presentation to the monitoring software (or prior to transmission over a T1/E1 link).

## Card Features

The following features are available on this DAG card. **Note:** Different firmware images may be required. Not all features are available on each firmware image. For further information on which feature is available in what firmware image, see Firmware images (page 19).

- ATM
- HDLC
- RAW E1/T1/J1
- IMA
- AAL2 and AAL5
- TERF

## Purpose of this User Guide

The purpose of this User Guide is to provide you with an understanding of the DAG 3.7T card architecture, functionality and to guide you through the following:

- Installing the card and associated software and firmware
- Configuring the card for your specific network requirements
- Running a data capture session
- Synchronizing clock time
- Data formats

You can also find additional information relating to functions and features of the DAG 3.7T card in the following documents which are available from the Support section of the Endace website at http://www.endace.com:

- *EDM04-01 DAG Software Installation Guide*
- *EDM04-03 dagflood User Manual*
- *EDM04-06 Daggen User Guide*
- *EDM04-08 Configuration and Status API Programming Guide*
- *EDM04-12 DAG 3.7T HDLC Filtering Guide*
- *EDM04-13 SAR API Programming Guide*
- *EDM04-18 IMA Host API Programming Guide*
- *EDM04-19 DAG Programming Guide*
- *EDM05-01 Time Distribution Server User Guide*
- *PN01-13 DAG Card Quick Start Guide*

This User Guide and the *EDM04-01 DAG Software Installation Guide* are also available in PDF format on the installation CD shipped with your DAG 3.7T card.

## System Requirements

### General

The minimum system requirements for the DAG 3.7T card are:

- A computer, with at least a Intel Xeon 1.8GHz or faster and a minimum of 1GB RAM.
- At least one free PCI 2.1 slot supporting 33MHz operation.
- Software distribution requires 60MB free space.
- For details of the supported operating systems, see one of the following documents:
  - *EDM04-01 DAG Software Installation Guide*
  - Current release notes - See the Documentation CD or the Endace support website at https://www.endace.com/support.

### Operating System

This document assumes you are installing the DAG 3.7T card in a computer which already has an operating system installed. To install refer to *EDM04-01 DAG Software Installation Guide*. All related documentation is included on the CD shipped with the DAG 3.7T card.

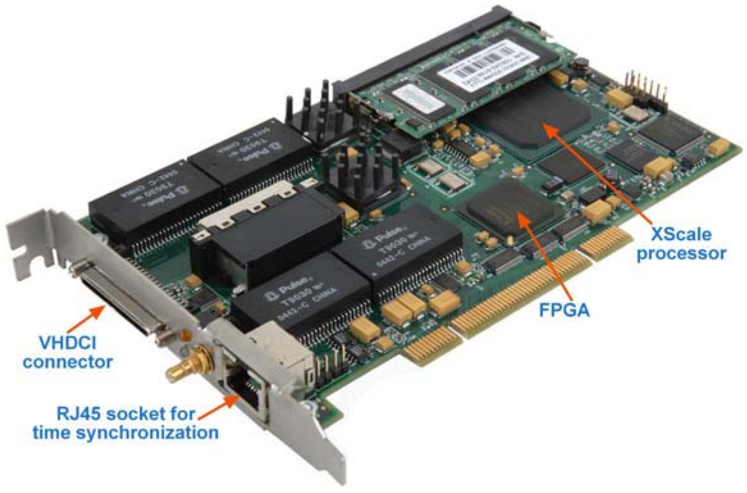
### Other Systems

For advice on using an operating system that is substantially different from any of those specified above, please contact Endace Customer Support at support@endace.com.

## Card Description

The DAG 3.7T cards are PCI bus cards designed for cell and packet capture and generation over IP networks. The key features of the card are:

- Support for 16 RJ-45 T1/E1 network interfaces in an external pod housing.
- A Spartan III FPGA supporting high-performance Endace firmware,
- An Intel 80321 XScale IO processor which supports AAL2/AAL5 reassembly or inverse multiplexing over ATM (IMA) and filtering services,
- Support for receiving and sending channelized, unchannelized, and fractional T1/E1, HDLC and non-HDLC data traffic,
- Support for data traffic filtering.



### Battery removal – don't do it!

**Removing the battery from a DAG card voids your warranty.**

Removing the battery from a DAG card will cause the loss of encryption key used to decode the DAG card's firmware. Once the encryption key is lost the DAG card must be returned to Endace for reprogramming.

The battery in this product is expected to last a minimum of 10 years.

**Caution**

Risk of explosion if the battery is replaced by an incorrect type.
Dispose of used batteries carefully.

## Card Architecture

T1 or E1 data is received on up to 16 x RJ-45 interfaces, and passes through line interface units. It then feeds immediately into the FPGA for deframing and demapping into ATM or HDLC frames.

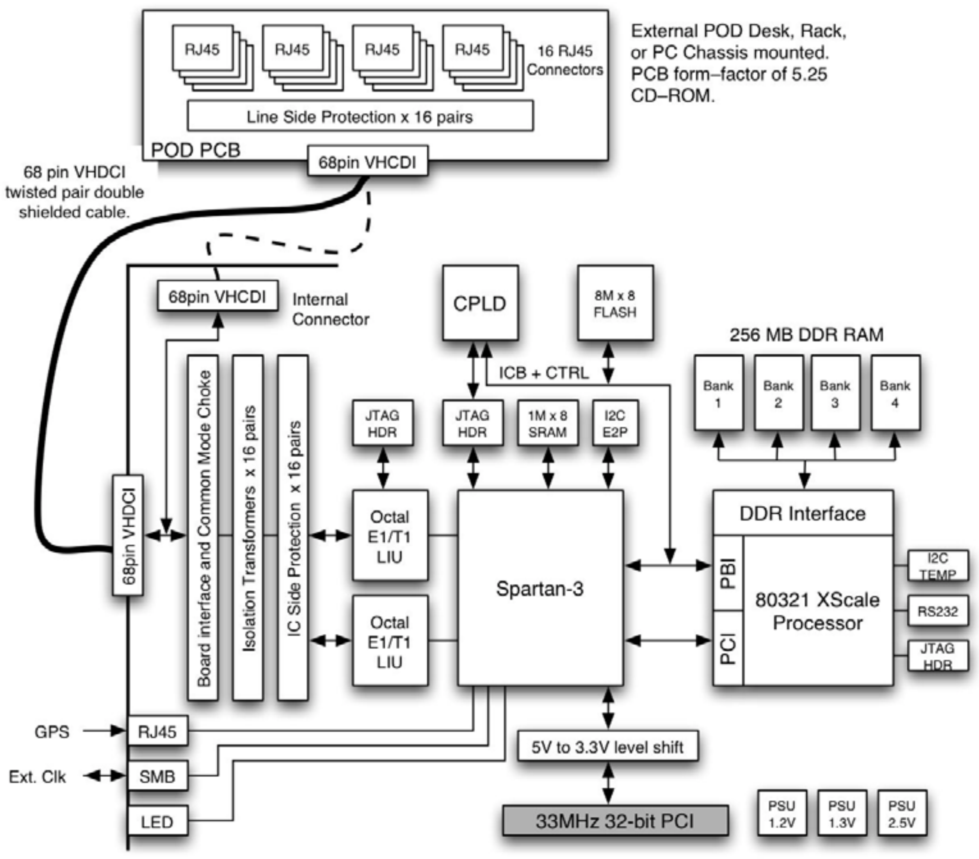
The FPGA contains a Packet processor and the DAG Universal Clock Kit (DUCK) timestamp engine. The DUCK provides high resolution per packet timestamps which can be accurately synchronized. Time stamped packet records are then stored in the lower FIFO.

**Note:** For further information on the time synchronization see Synchronizing Clock Time (page 65) later in this User Guide.

An Intel 80321 XScale processor is logically located next to the main FPGA. The XScale processor provides the facility to pre-process data before it is presented to the host, or before being transmitted over an E1/T1 link. It can also facilitate hostless operation via an embedded Linux kernel.

The main FPGA can route packets to either the XScale processor before routing onto the host, or directly to the host via the PCI port.

The diagram below shows the card's major components and the flow of data.



## Line Types

It is important that you understand the physical characteristics of the network to which you want to connect. If your configuration settings do not match your network, the DAG 3.7T card will not function as expected.

### Overview

Endace DAG 3.7T card provides the means to transfer data at the full speed of the network into the memory of the host computer, with zero packet loss guaranteed in even worst-case conditions. Further, unlike a Network Interface Card (NIC), Endace products actively manage the movement of network data into memory while only consuming a minimal amount of the host computer's resources. The full attention of the CPU remains focused on the analysis of incoming data without a constant stream of interruptions as new packets arrive from the network. For a busy network link, this feature has a turbo-charging effect similar to that of adding a second CPU to the system.

The DAG 3.7T is a Network Monitoring Interface Card specifically designed to perform high efficiency monitoring and transmission with precision timestamping capability on up to sixteen T1/E1 network links.

The flexibility provided by the Exar chips means that the card will accept a wide range of settings. However if they are not the correct settings for your network the card will not function as expected.

**Note:** If you are unsure about which of the options listed below to apply to your network, please contact your Network Administrator for further information.

### Supported Options

The line characteristics supported by the DAG 3.7T card are described below.

#### Line Type:

- **E1:** European digital standard 2 Mbps,
- **E1 CRC:** E1 with cyclic redundancy check,
- **T1**: North American digital standard 1.544 Mbps,
- **T1 SF:** Super frame, (also called D4 framing). An SF is 12 frames long,
- **T1 ESF**: Extended super frame, (also called D5 framing), includes CRC and bandwidth for a data link channel. An ESF is 24 frames long.

#### Encoding Type:

- **B8ZS:** Bipolar with Eight Zero Substitution (T1 only)/**HDB3** Hi-density Bipolar Three zeros(E1 only),
- **AMI:** Alternate Mark Inversion.

#### Cable Termination Types:

- Externally terminated,
- 75Ω unbalanced coaxial cable (E1 only),
- 100Ω balanced twisted pair (T1 only),
- 120Ω balanced twisted pair (E1 only).

#### Signal Attenuation:

- Maximum receiver gain of 36dB for T1.
- Maximum receiver gain of 43dB for E1.

## Extended Functions

The DAG 3.7T card supports the following extended functions:

- AAL2/AAL5 segmentation and  reassembly
- Inverse Multiplexing ATM (IMA)
- HDLC Filtering (see *EDM04-12 DAG 3.7T HDLC Filtering Guide*)

These functions are described in more detail in Using your DAG card to capture data (page 49) later in this User Guide and also in the following documents available from Endace Customer Support at support@endace.com:

## Introduction

A DAG 3.7T card can be installed in any free PCI slot. It is 5V tolerant and operates only in 32-bit 33MHz PCI mode.

If you install the card into a slot that is rated for higher speeds it will cause the bus to automatically change to 33MHz. This will also affect any other devices which may be sharing the bus.

You can run multiple DAG 3.7T cards on one bus. By default, the DAG driver supports up to four DAG cards in one system.

## DAG Software package

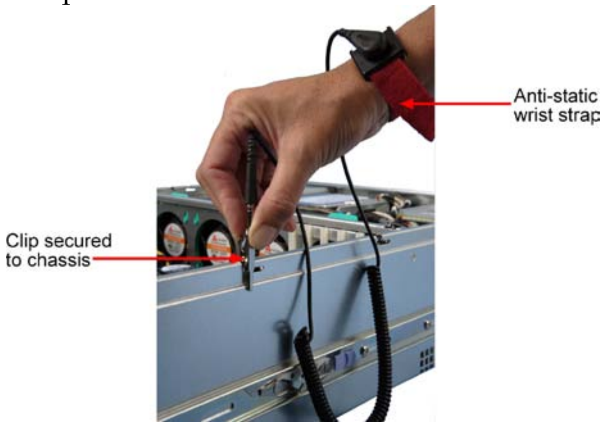
The latest DAG Software package must be installed before you install the DAG 3.7T card itself. See *EDM04-01 DAG Software Installation Guide*, which is included on the CD shipped with the DAG 3.7T card.

## Inserting the DAG Card

**Caution:**

It is very important to protect both the computer and the DAG 3.7T card from damage by electro-static discharge (ESD). Failure to do so could cause damage to components and subsequently cause the card to partially or completely fail.

1. Turn power to the computer OFF.
2. Remove the PCI bus slot screw and cover.
3. Using an approved ESD protection device attach the end with the strap to your wrist and pull or clip firmly so there is firm contact with your wrist.
4. Securely attach the clip on the other end of the strap to a solid metal area on the computer chassis as shown below.



5. Insert the DAG 3.7T card into PCI bus slot ensuring it is firmly seated.
6. If this DAG card requires an external power supply, complete the following steps:
   a. Connect the supplied (or equivalent) power cable to the external power connector on the DAG card.
   b. Connect the cable to the appropriate power connector on your server's power supply unit.
7. Check the free end of the card fits securely into the card-end bracket that supports the weight of the card.
8. Secure the card with the bus slot cover screw.
9. Turn power to the computer ON.
10. Ensure the blue (FPGA successfully programmed) LED on the DAG card illuminates.

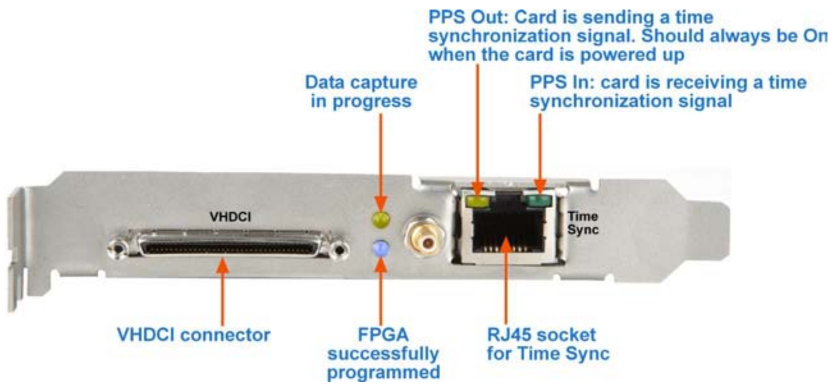
## Port Connectors

Before you begin to configure the DAG card it is important to understand the function of the various LEDs associated with the card, as well as the sockets on the PCI bracket.

There is an 8-pin RJ-45 PPS input socket located below the SFP connectors on the PCI bracket. This is available for connection to an external time synchronization source only.

**Caution:**

Never connect an Ethernet network or telephone line to the RJ-45 PPS input socket.


PPS Out: Card is sending a time synchronization signal. Should always be On when the card is powered up
Data capture in progress
PPS In: card is receiving a time synchronization signal
VHDCI
Time Sync
VHDCI connector
FPGA successfully programmed
RJ45 socket for Time Sync

## External pod housing

There are two forms of external Pod housing that are available:

- an external Pod case - housed in a 5.25 inch drive bay housing. For further details see External Pod (page 9).
- a Pod rackmount chassis - a 1U, 19 in rack housing. For further details see Pod Chassis (page 10).

## External Pod

**Note:** You can connect only one Pod to the DAG 3.7T card at any one time.

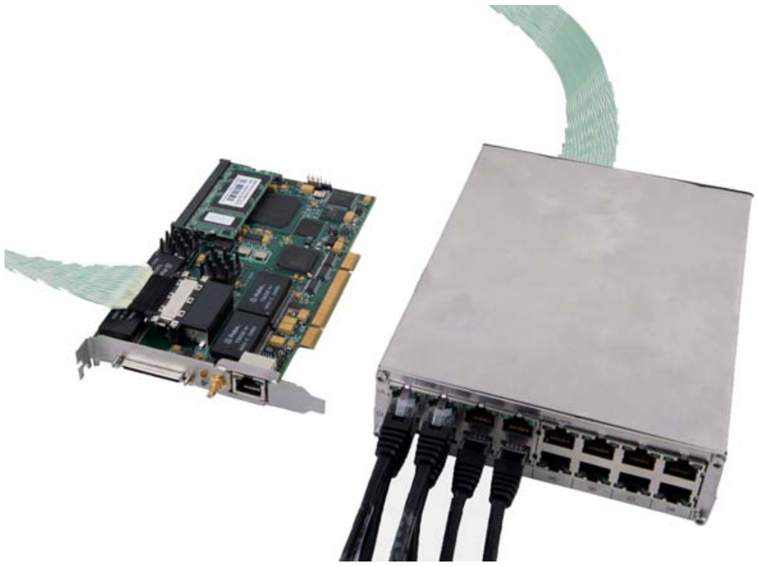
You can mount the DAG 3.7T Pod either:

- internally in a spare 5.25 inch drive bay in the computer chassis, or
  Use the supplied VHDCI ribbon cable.
- sit it separately outside of the computer chassis.
  Use a shielded VHDCI ribbon cable rather than the supplied ribbon cable.

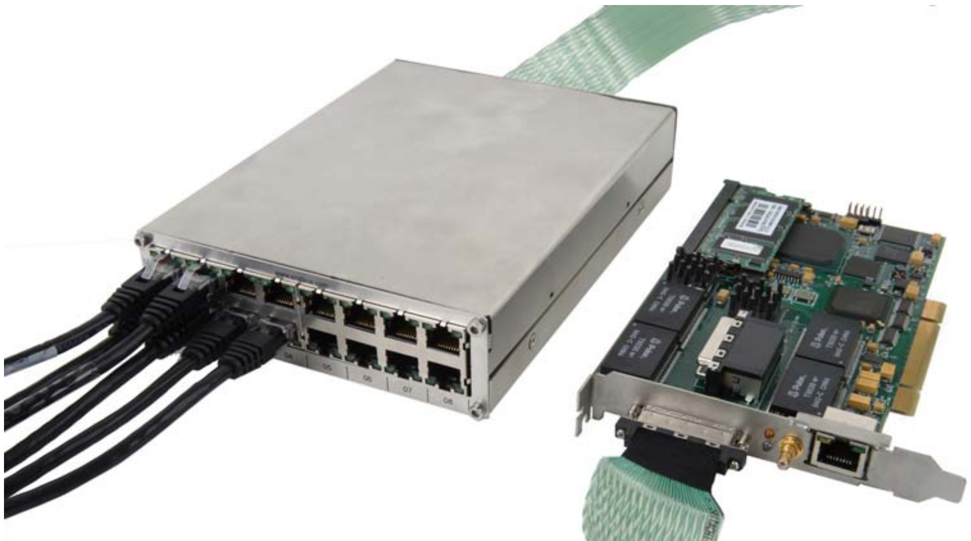
The Pod and the DAG 3.7T card connect via a VHDCI cable which is supplied with the Pod. The DAG 3.7T card has one VHDCI connector located on the PCI bracket and another on the card itself. The Pod has one VHDCI connector located on the rear of the casing.

### Connecting the external Pod to the computer

If the Pod is mounted internally in the computer chassis you need to use the VHDCI connector located on the card itself as shown in the picture below.

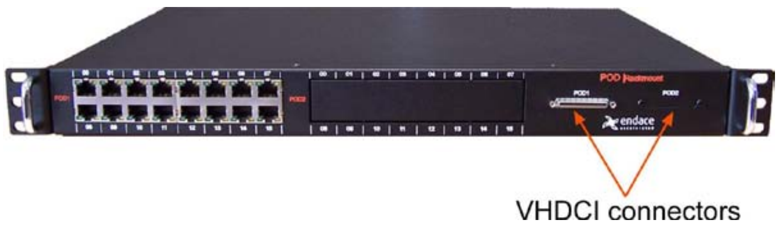


If the Pod is mounted external to the computer chassis you need to use the connector located on the card PCI bracket as shown in the picture below.



## Pod rackmount chassis

This section describes how to install the Pod rackmount chassis ready for connecting to the monitored network.



#### Additional options

For details on how to:

- add a second Pod PCB into a Pod rackmount chassis, see Adding a second Pod PCB to the Pod rackmount chassis (page 14).
- change the VHDCI connector location, see Changing VHDCI connector location (page 18).

### Before you begin

Inspect the contents of the Pod rackmount chassis kit and ensure you have the following items:

- 1 x Sliding rail kit
- 1 x Pod rackmount chassis  (PodRMount-37P1 has one Pod  or PodRMount-37P2 has two Pods)

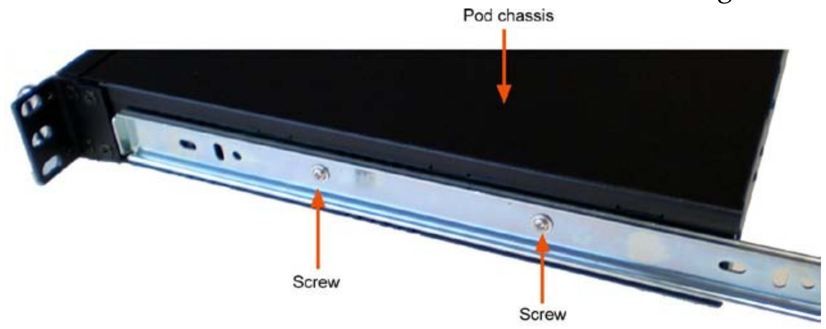
### Installing the Pod rackmount chassis

#### Step 1: Separate the rail assembly

1. Slide inner rail out to full extension.
2. Press the release lever and slide inner rail from outer rail.
3. Repeat the following for the second rail assembly.

#### Step 2: Attach rail to Pod chassis

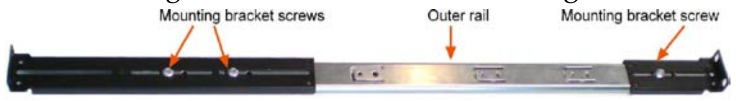
1. Attach the inner rail to the Pod rackmount chassis using the two screws (supplied).



2. Repeat for the other side.
   **Note:** Ensure both release levers are towards the rear of the Pod rackmount chassis.

#### Step 3: Attach rail assembly to rack

1. Measure the distance from the front to the rear of the rack and compare to the length of the outer rail.
2. Ensure both mounting brackets are attached to the outer rail.
3. Adjust the length of the outer rail to match the rack depth by adjusting the screws on the mounting brackets. Leave the mounting bracket screws un-tightened.



4. Attach the front and rear mounting brackets to the rack using supplied screws.
5. Adjust the mounting brackets attachments until the outer rail fits in the rack and then tighten the screws.
6. Tightened all screws on the mounting brackets.
7. Repeat for the other outer rail.

#### Step 4: Mount the Pod rackmount chassis in the rack

1. Slide the Pod rackmount chassis inner rails into the rack into the mounted outer rails until you hear the release/locking lever operate.
2. Press in the release/locking lever on both sides and continue to slide the Pod rackmount chassis into the rack until it will not slide any further.

### Cable wiring

The RJ45 cables for connecting the network interfaces to the Pod are not supplied with the DAG 3.7T card or the Pod. You must source these yourself.

If you have an Endace Pod, you are able to use standard E1/T1 cables available from your local electronic stockist. If you do not have an Endace Pod you will need to make the cables up yourself to match the Pod pinouts shown in the following tables.

**Note:** You can identify the standard Endace Pod by the web address "www.endace.com" written on the front of the casing.

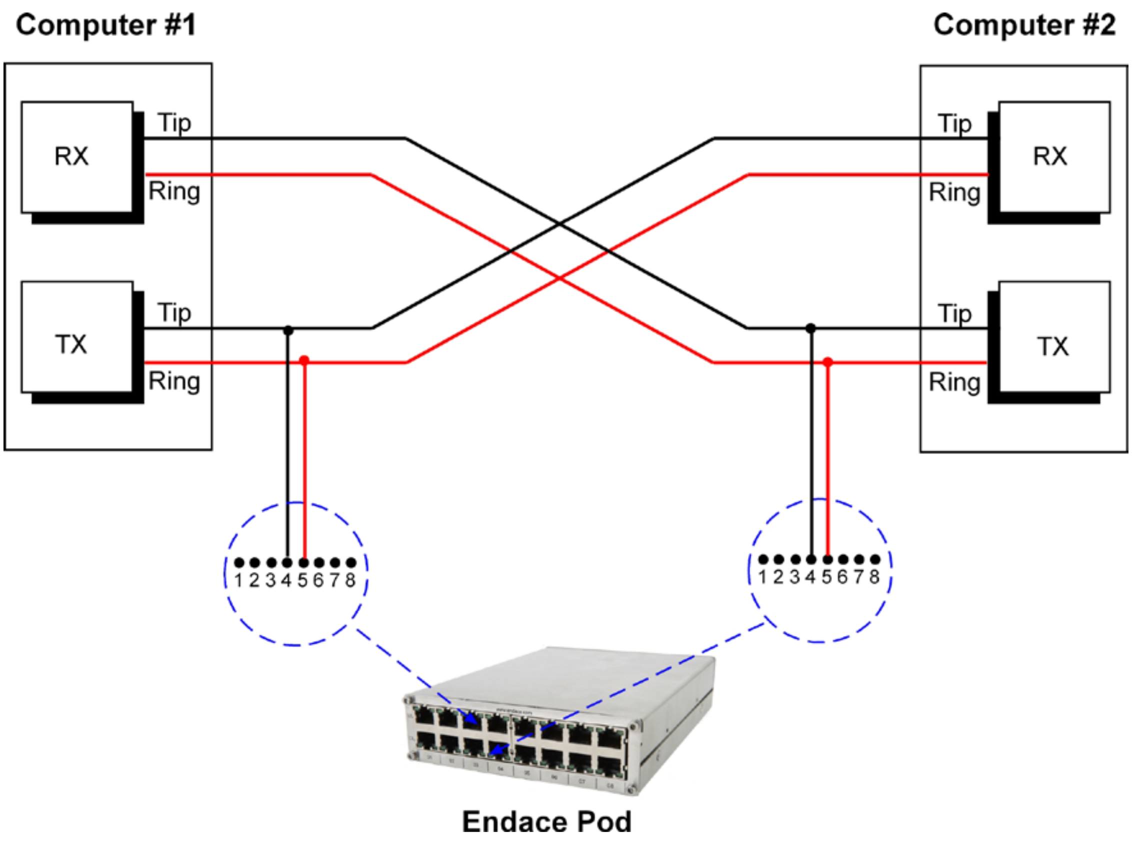
The physical pinouts of the RJ-45 connectors for both Endace and other pods are shown below:

| Endace Pod | |
|---|---|
| 1 | TX Tip |
| 2 | TX Ring |
| 3 | |
| 4 | RX Tip |
| 5 | RX Ring |
| 6 | |
| 7 | |
| 8 | |

| Other Pod | |
|---|---|
| 1 | |
| 2 | |
| 3 | TX Ring |
| 4 | TX Tip |
| 5 | RX Ring |
| 6 | RX Tip |
| 7 | |
| 8 | |

### Connecting to the Network

Once you have connected the interfaces to the Pod you must connect the interfaces to the network via a tap as shown in the diagram below:



You must use a separate tap for each interface.

When you have connected the Pod to the DAG 3.7T and the network interfaces to the network, you must configure the card for your specific requirements. This process is described next in Configuring the DAG Card (page 19).

**Note:** For further information about using taps to connect to the network, please consult your network administrator.

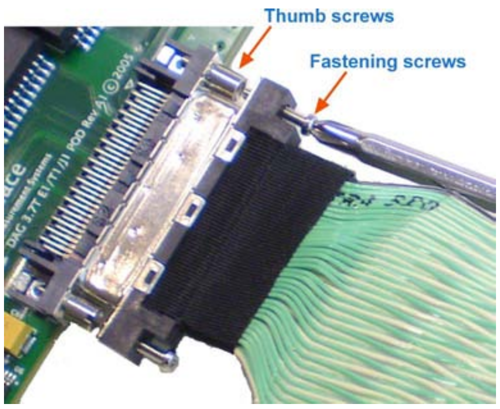
### Adding a second Pod PCB to the Pod rackmount chassis

The following describes how to add a second Pod PCB into the Pod rackmount chassis.

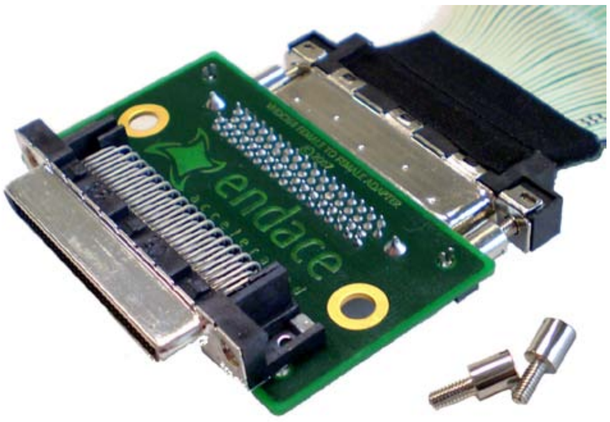
Note: For instructions on removing a Pod PCB from the external Pod case, see Removing a Pod PCB from an External Pod case (page 16)

1. Remove the top cover from the Pod rackmount chassis.
2. Remove the front blanking plate. Break the metal tabs holding the blanking plate in place.

3. Attach the VHDCI ribbon cable to the Pod PCB.
   a. Attach thumb screws to connector on Pod PCB.

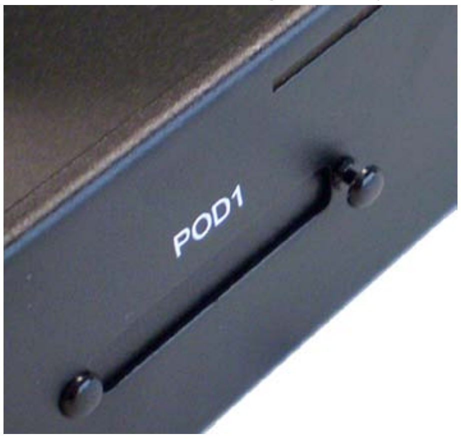


   b. Connect VHDCI ribbon cable to Pod PCB.
   c. Fasten the VHDCI ribbon cable to Pod PCB using the supplied screws.
4. Attach the other end of the VHDCI ribbon cable to the VHDCI PCB using the supplied screws.

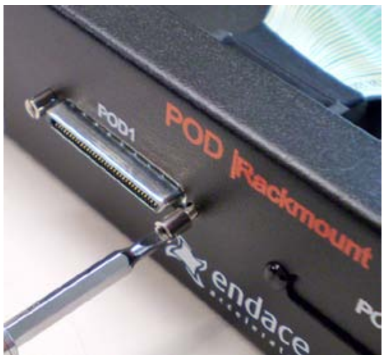


5. Remove thumb screws and retain.
6. Select which Pod 2 connector location you want to use (front or back).

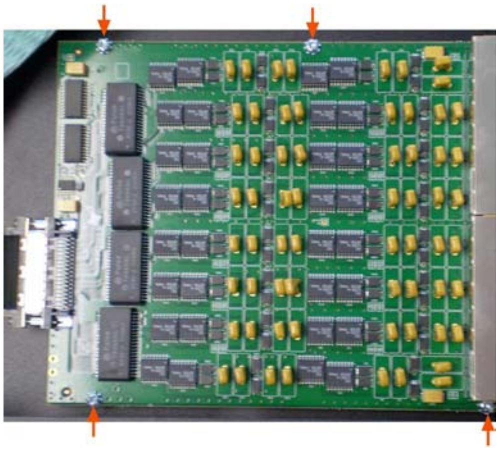
7. Remove the blanking plate by pulling out the center of the plastic rivet.



8. Mount the VHDCI PCB in the required location using the existing thumb screws.



9. Mount the Pod PCB in the Pod chassis using four screws (supplied).



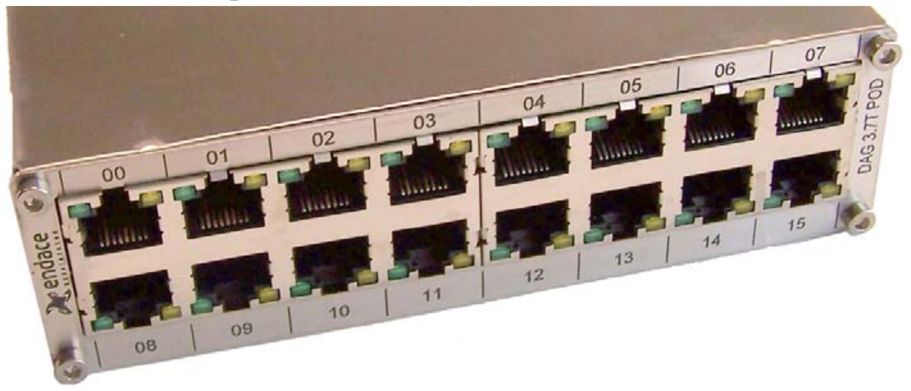
10. Replace the top cover (two screws).

### Removing a Pod PCB from an External Pod case

The following steps explain how to remove the Pod PCB from the External Pod case.

1. Remove the VHDCI cable connector from the back of the External Pod case (two thumb screws - keep).

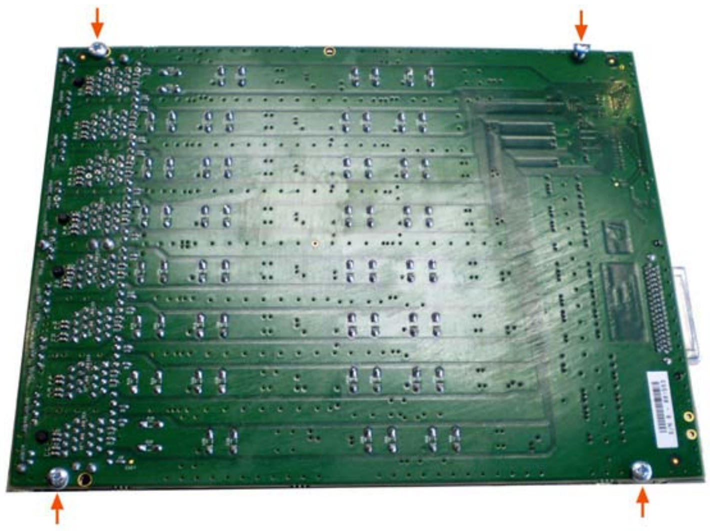
2. Remove the front panel (four screws).



3. Remove screws from the sides of the External Pod case (two screws per side)

4. Separate the two halves of case starting at the front and using moderate force.

5. Remove the Pod PCB from the case (four screws).

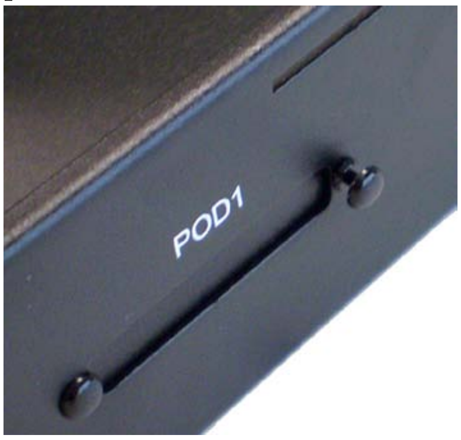


### Changing VHDCI connector location

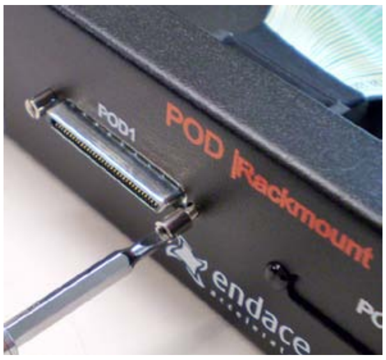
The VHDCI connector exits the Pod rackmount chassis from the front by default. If required you can change the exit location to the rear.

The following steps describe how to change the VHDCI connector exit location:

1. Select which connector location you want to use.
2. Remove the blanking plate by pulling out the center of each plastic rivet - retain the plastic rivets.



3. Mount the VHDCI PCB in the required location.



4. Mount the blanking plate in the unused locations using the plastic rivets.

Notes:

- Ensure you use the correct VHDCI connector location, i.e. Pod 1 for the connector from the Pod 1 PCB, etc.
- Take care with the VHDCI ribbon cable.

# Configuring the DAG card

## Introduction

Configuring the DAG 3.7T card ready for capturing data requires the following steps:

- Setting up the FPGA (page 20)
- Preparing the DAG card for use (page 22)
- Configuring the DAG Card (page 23)
- Viewing the DAG card statistics (page 34)

Once the DAG 3.7T is configured you can start capturing data, see Using your DAG card to capture data (page 49) for details on capturing data.

### Before configuring the DAG card

Before configuring the FPGA, you should ensure that:

- `dagmem` has been run and memory allocated to each installed DAG card.
- `dagload` has been run so that all DAG drivers have been installed.

Refer to the *Installing the drivers* section for the required Operating system in *EDM04-01 DAG Software Installation Guide* for the further details.

### Firmware images

The following lists the features available on each firmware image available on this DAG card.

| FPGA image (Software version string) | RAW E1/T1/J1 | IMA | AAL2 & AAL5 | ATM | HDLC | TERF |
|---|---|---|---|---|---|---|
| `dag37tpci_erf.bit` (`dag37tpci_erf_pci...`) | ✓ | | | | ✓ | ✓ |
| `dag37tpci_mixed.bit` (`dag37tpci_erf-mixed_pci...`) | ✓ | ✓ | ✓ | ✓ | ✓ | |
| `dag37tpci_erf-atm.bit` (`dag37tpci_erf-atm_pci...`) | | ✓ | ✓ | ✓ | | ✓ |

The software version strings are displayed in the `dagconfig` output and when using the `dagrom -x` command. They include a version number and creation date.

## Setting up the FPGA

All DAG cards have at least one Field-Programmable Gate Array (FPGA). The FPGA contains the firmware for the DAG card. The firmware defines how the DAG card operates when capturing data and contains the specific configuration.

**Note:** Some DAG cards have multiple FPGA's.

For each FPGA there are two firmware images:

- a factory image - contains fixed basic functionality for operating the DAG card.
- a user image - contains an upgradable version of the DAG card firmware. Additional functionality for the DAG card is available via the user image. Different user images may be available with different functionality, i.e. TERF, DSM etc.

Firmware images are loaded into DAG card flash ROM in the factory. The image is programmed into the FPGA each time the DAG card is powered up. The user image can then be programmed into the FPGA either manually or via a script.

### Programming the FPGA

Before configuring the DAG card for capture, you must load and program the DAG card with the appropriate FPGA image.

**Note:** For information about the `dagrom` options, see dagrom (page 21).

- For High Data Lin Control (HDLC) capture, use:

```
dagrom –rvp –d0 –f xilinx/dag37tpci-hdlc-erf.bit
```

- For Asynchronous Transfer Mode (ATM) capture, use:

```
dagrom –rvp –d0 –f xilinx/dag37tpci-atm-erf.bit
```

- For ATM **and** HDLC capture, use:

```
dagrom –rvp –d0 –f xilinx/dag37tpci-mixed-erf.bit
```

where "0" is the device number of the DAG card you wish to capture data from

**Note:** The `mixed` FPGA image does not support transmit.

## dagrom

`dagrom` is a software utility that enables you to configure the FPGA on Endace DAG cards. The following is a list of options available in `dagrom`.

| Option | Description |
|---|---|
| `-a,--alternate-half` | Use alternate (stable) half. [Default is current half.] Factory / User. |
| `-A,--entire-rom` | Entire ROM. [Default is current half only.] |
| `-b,--swid-rom-check` | Check if there is a SWID on the ROM. |
| `-c,--cpu-region <region>` | Access CPU region: c=copro, b=boot, k=kernel, f=filesystem. |
| `--continue` | Continue on erase error. |
| `-d,--device <device>` | DAG device to use. |
| `-e,--erase` | Erase ROM. [Default is read.] |
| `-F,--disable-cfi-fast` | Disable fast program option for CFI mode. |
| `-f,--file <filename>` | File to be read when programming ROM. There are multiple FGPA images per DAG card, covering the different versions, ERF, TERF DSM etc. |
| `--force` | Force loading firmware. Dangerous. |
| `-g,--rom-number <rom>` | Access specified ROM controller. [Default is 0.] |
| `-h,--help`<br>`-?,--usage` | This page. |
| `-i,--halt-ixp` | Halt the embedded IXP Processor (DAG 7.1S only). |
| `--image-table-fpga <image table fpga>` | Specify the Power On image selection table FPGA number |
| `--image-table-image <image table image>` | Specify the Power On image selection table Image number |
| `-j,--swid-rom-check-key <key>` | Check the ROM SWID key with the one supplied. |
| `-l,--hold-bus` | Hold PBI bus from XScale (DAG 3.7T only). |
| `-m,--swid-key <key>` | Hexadecimal key for writing the Software ID (aka SWID). |
| `-o,--swid-rom-read` | Read SWID from ROM. |
| `-p,--program-current` | Program current User 1 Xilinx image into FPGA. |
| `-q,--image-number <image number>` | Specify the image number to write or to program the card.[0 - 3]. 0 factory image, 1 user image 1, 2 user image 2, 3 user image 3. (7.5G2/G4 only) |
| `--swid-write <swid>` | Write given SWID. The key must be supplied with the -m option, requires a valid running XScale ROM Image. (3.7T, 3.7D, 3.8S and 7.1S only) |
| `-r,--reprogram` | Reprogram ROM (may imply erase and write). |
| `--reset-method <reprogram method>` | Specify the method to reprogram the card.[1.Ringo 2.George 3.Dave] |
| `-s,--swid-rom-write <swid>` | Write given SWID to ROM. The key must be supplied with the -m option. |
| `-t,--swid-read-bytes <bytes>` | Read <bytes> of SWID, requires a valid running XScale ROM image (3.7T only) |
| `-u,--swid-erase` | Erase SWID from ROM. |
| `--unknown` | Force loading firmware. Dangerous. |
| `-v,--verbose` | Increase verbosity. |
| `-V,--version` | Display version information. |
| `-w,--write` | Write ROM (implies erase). [Default is read.] |
| `--write-out <filename>` | Write the contents of the ROM to a file. |
| `-x,--list-revisions` | Display Xilinx revision strings (the default if no arguments are given). |
| `-y,--verify` | Verify write to ROM. |
| `-z,--zero` | Zero ROM. [Default is read.] |

All commands apply to the current image portion of the ROM, unless one of the options `-a`, `-A`, `-c` is specified.

**Note:** Not all commands are supported by all DAG cards.

To view the FPGA image revision strings, type the following:

```
dagrom -d0 -x
```

Output:

```
user:          dag37tpci_erf-atm_v2_7 3s1500fg456 2006/10/11 15:47:06 (active)
factory:  dag37tpci_erf_v2_2 3s1500fg456 2005/03/04 16:46:17
```

### Loading new firmware images onto a DAG Card

New DAG card FPGA images are released regularly by Endace as part of software packages. They can be downloaded from the Endace website at https://www.endace.com/support.

Endace recommends you use the `dagrom -r` command when loading images from the computer to the ROM on the DAG card.

The `-r` option invokes a comparison of images on the computer and in the DAG card. Newer versions are automatically loaded onto the DAG card and programmed into the FPGA. See dagrom (page 21). This eliminates unnecessary reprogramming of the ROM and extends its life.

## Preparing the DAG card for use

Before configuring the DAG 3.7T card you must run the following `dagconfig` command to set the default parameters in the DAG card. This ensures the DAG 3.7T card functions correctly once you begin capturing data.

**Note:** Ensure you run this command each time the FPGA is reprogrammed.

```
dagconfig -d0 default
```

The current DAG 3.7T configuration displays and the firmware is verified as correctly loaded. See dagconfig (page 32) for more information.

# Configuring the DAG card

## Display Current Configuration

Once you have loaded the FPGA image you should run the `dagconfig` tool without arguments to display the current card configuration and verify the firmware has been loaded correctly.

To display the current DAG card configuration, type the following:

`dagconfig` –d0

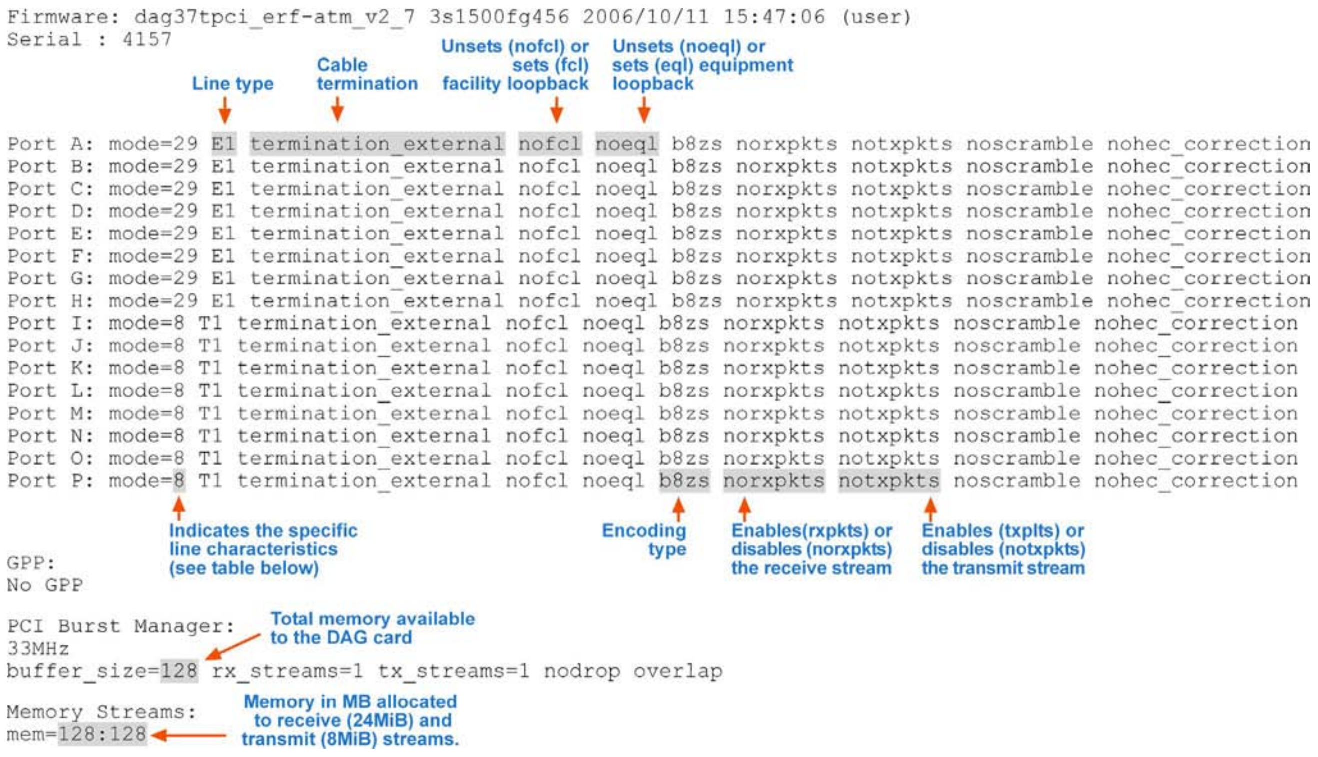
(where "0" is the device number of the DAG card you wish to capture data from).
A description of available tokens follows.

**Note:** Not all tokens displayed in the following diagram.

`dagthree` has been depreciated from DAG Software release 3.2 onwards. It has been replaced with `dagconfig`. Both are still valid. Endace recommends that new customer use `dagconfig`.

### Display Current Configuration

An example of a `dagconfig` output of a erf / atm image loaded is shown below:



```
Firmware: dag37tpci_erf-atm_v2_7 3s1500fg456 2006/10/11 15:47:06 (user)
Serial : 4157

Port A: mode=29 E1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port B: mode=29 E1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port C: mode=29 E1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port D: mode=29 E1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port E: mode=29 E1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port F: mode=29 E1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port G: mode=29 E1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port H: mode=29 E1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port I: mode=8 T1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port J: mode=8 T1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port K: mode=8 T1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port L: mode=8 T1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port M: mode=8 T1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port N: mode=8 T1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port O: mode=8 T1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction
Port P: mode=8 T1 termination_external nofcl noeql b8zs norxpkts notxpkts noscramble nohec_correction

GPP:
No GPP

PCI Burst Manager:
33MHz
buffer_size=128 rx_streams=1 tx_streams=1 nodrop overlap

Memory Streams:
mem=128:128
```

Line type
Cable termination
Unsets (nofcl) or sets (fcl) facility loopback
Unsets (noeql) or sets (eql) equipment loopback
Indicates the specific line characteristics (see table below)
Encoding type
Enables(rxpkts) or disables (norxpkts) the receive stream
Enables (txplts) or disables (notxpkts) the transmit stream
Total memory available to the DAG card
Memory in MB allocated to receive (24MiB) and transmit (8MiB) streams.

**Note**: The default configuration displays for all 16 links even if there is no interface physically connected to the Pod.

Returning the card to the default configuration means that all the links will have the same settings. You then only need to re-configure those you want to change from the default.

## Configuring the Links

### Overview

Each connected link on the DAG 3.7T Pod must be configured to the requirements of the tapped network. For each link complete the following steps to configure the link.

**Note:** Each link represents a network interface on the 3.7T Pod.

1. Identify network characteristics
2. Choose appropriate mode
3. Configure the link + examples

### 1. Identify network characteristics

To find out what kind of mode connection to use for your network you must know the physical characteristics of your network. The DAG 3.7T card supports the following options:

**Line Types:**

- **E1:** European digital standard 2 Mbps,
- **E1 CRC:** E1 with cyclic redundancy check,
- **T1**: North American digital standard 1.544 Mbps,
- **T1 SF:** Super frame, (also called D4 framing). An SF is 12 frames long,
- **T1 ESF:** Extended super frame, (also called D5 framing), includes CRC and bandwidth for a data link channel. An ESF is 24 frames long.

**Encoding Types:**

- **B8ZS:** Bipolar with Eight Zero Substitution (T1 only)/**HDB3** Hi-density Bipolar Three zeros(E1 only),
- **AMI:** Alternate Mark Inversion.

**Cable Termination Types:**

- Externally terminated,
- 75Ω unbalanced coaxial cable (E1 only),
- 100Ω balanced twisted pair (T1 only),
- 120Ω balanced twisted pair (E1 only).

**Signal Attenuation:**

- Maximum receiver gain of 36dB for T1.
- Maximum receiver gain of 43dB for E1.

## 2. Choose appropriate modes

The table below describes the different modes corresponding to line characteristics supported by the DAG 3.7T card. Select the required mode for each link.

**Note:** Ensure that the mode you select matches the physical characteristics of the network to which you want to connect.

| Mode | Type | Tx LBO | Cable Termination | Coding |
|---|---|---|---|---|
| 0 | T1 Long Haul/36dB | 0dB | 100Ω/ TP | B8ZS |
| 1 | T1 Long Haul/36dB | -7.5dB | 100Ω/ TP | B8ZS |
| 2 | T1 Long Haul/36dB | -15dB | 100Ω/ TP | B8ZS |
| 3 | T1 Long Haul/36dB | -22.5dB | 100Ω/ TP | B8ZS |
| 4 | T1 Long Haul/45dB | 0dB | 100Ω/ TP | B8ZS |
| 5 | T1 Long Haul/45dB | -7.5dB | 100Ω/ TP | B8ZS |
| 6 | T1 Long Haul/45dB | -15dB | 100Ω/ TP | B8ZS |
| 7 | T1 Long Haul/45dB | -22.5dB | 100Ω/ TP | B8ZS |
| 8 | T1 Short Haul/15dB | 0-133 ft./ 0.6dB | 100Ω/ TP | B8ZS |
| 9 | T1 Short Haul/15dB | 133-266 ft./ 1.2dB | 100Ω/ TP | B8ZS |
| 10 | T1 Short Haul/15dB | 266-399 ft./ 1.8dB | 100Ω/ TP | B8ZS |
| 11 | T1 Short Haul/15dB | 399-533 ft./ 2.4dB | 100Ω/ TP | B8ZS |
| 12 | T1 Short Haul/15dB | 533-655 ft./ 3.0dB | 100Ω/ TP | B8ZS |
| 13 | T1 Short Haul/15dB | Arbitrary Pulse | 100Ω/ TP | B8ZS |
| 14 | T1 Gain Mode/29dB | 0-133 ft./ 0.6dB | 100Ω/ TP | B8ZS |
| 15 | T1 Gain Mode/29dB | 133-266 ft./ 1.2dB | 100Ω/ TP | B8ZS |
| 16 | T1 Gain Mode/29dB | 266-399 ft./ 1.8dB | 100Ω/ TP | B8ZS |
| 17 | T1 Gain Mode/29dB | 399-533 ft./ 2.4dB | 100Ω/ TP | B8ZS |
| 18 | T1 Gain Mode/29dB | 533-655 ft./ 3.0dB | 100Ω/ TP | B8ZS |
| 19 | T1 Gain Mode/29dB | Arbitrary Pulse | 100Ω/ TP | B8ZS |
| 20 | T1 Gain Mode/29dB | 0dB | 100Ω/ TP | B8ZS |
| 21 | T1 Gain Mode/29dB | -7.5dB | 100Ω/ TP | B8ZS |
| 22 | T1 Gain Mode/29dB | -15dB | 100Ω/ TP | B8ZS |
| 23 | T1 Gain Mode/29dB | -22.5dB | 100Ω/ TP | B8ZS |
| 24 | E1 Long Haul/36dB | ITU G.703/Arbitrary | 75Ω/ coax | HDB3 |
| 25 | E1 Long Haul/36dB | ITU G.703/Arbitrary | 120Ω/ TP | HDB3 |
| 26 | E1 Long Haul/43dB | ITU G.703/Arbitrary | 75Ω/ coax | HDB3 |
| 27 | E1 Long Haul/43dB | ITU G.703/Arbitrary | 120Ω/ TP | HDB3 |
| 28 | E1 Short Haul/15dB | ITU G.703/Arbitrary | 75Ω/ coax | HDB3 |
| 29 | E1 Short Haull/15dB | ITU G.703/Arbitrary | 120Ω/ TP | HDB3 |
| 30 | E1 Gain Mode | ITU G.703/Arbitrary | 75Ω/ coax | HDB3 |
| 31 | E1 Gain Mode | ITU G.703/Arbitrary | 120Ω/ TP | HDB3 |

### 3. Configure the link

To set the desired mode of operation use the appropriate `dagconfig` commands.
For more information on the various options available see `dagconfig` output explained.

You must set **all** the following options for each link or default setting is retained.

- Mode
- Line type
- Termination
- Coding

You can:

- change multiple settings at the same time and
- configure more than one link at the same time.

The new settings are applied in the order in which you specify them.

e.g. to set the DAG 3.7T card in slot 0, to link 1 to mode = 10, T1 line type, 100Ω twisted pair cable and B8ZS/HDB3 encoding use the following commands:

```
dagconfig -d0 link=1 mode=10 T1 term100 B8ZS
```

**Examples**

If you do not specify a link the changes are applied to all links. In the example below, all links will be changed to mode 29, with 100Ω termination:

```
dagconfig -d0 mode=29 term100
```

To change link 1 to T1 using mode 8 and eql , with 100Ω termination use:

```
dagconfig -d0 link=1 mode=8 eql term100
```

To change link 2 to mode 29 and link 3 to mode 29 using B8ZS encoding but receive only, use:

```
dagconfig -d0 link=2 mode=29 link=3 mode=29 notxpkts B8ZS
```

**Using default**

Using the default command, resets any altered DAG card settings to the factory default settings. You can not return individual links to default setting.

## dagconfig tokens explained

The DAG 3.7T card now uses `dagconfig` instead of `dagthree`. The tokens listed below can be used with `dagconfig`.

### align64

Sets whether the generated packets are 64-bit aligned (`align64`) or 32-bit aligned (`noalign64`) before being received by the host.

Example

```
  dagconfig align64
  dagconfig noalign64
```

### buffer_size

The buffer size=*n*MB indicates that a total of *n* MB of memory have been allocated to the DAG card in total. Memory allocation occurs when the `dagmem` driver is loaded at boot time. See *EDM04-01 DAG Software Installation Guide* for details on how to allocate memory.

### crc

Indicates that this DAG card is set to perform a Cyclic Redundancy Check (CRC) on the receive stream.

Not configurable.

### default

The `default` command initializes the DAG card configuration and sets all settings to default values. The command also resets the DAG card configuration back to its default state.

**Note:** When you run `dagconfig -d0 default` the `dagclock` inputs and outputs are also reset to defaults.

Example

```
  dagconfig -d0 default
```

### drop

Details the number of packets dropped during current capture session. Resets to 0 when the session restarted. Indication only can not be changed.

Example

The following shows that 15 packets have been dropped in the current session:

```
  drop=15
```

### e1 / e1_crc / e1_unframed

Sets the E1 line characteristics for the ports in the range A to H or I to P.

### eql/noeql

Sets or unsets equipment loopback. For testing set to `eql` mode and normal operation set to `noeql` mode.

**Note:** `eql` mode loops transmit data from the host back to the PCI bus.

Example

```
  dagconfig eql
  dagconfig noeql
```

### Error

Turns on or off the error reporting for this DAG card.

Example

```
dagconfig error
dagconfig noerror
```

### fcl/nofcl

**Note:** Sets or unsets Facility loop back. For testing set to `fcl` mode and normal operation set to `nofcl` mode.

FCL retransmits the data received and also send it to the host.

Example

```
dagconfig fcl
dagconfig nofcl
```

### HDLC

Tells the card to process HDLC packets.

Example

```
dagconfig hdlc
```

### mem

You can split the DAG card's allocated memory between the receive and transmit stream buffers to suit your own requirements. The split is displayed as a ratio as shown below:

```
mem=X:Y
```

where:
`X` is the memory allocated in MB to the `rx` stream
`Y` is the memory allocated in MB to the `tx` stream.

If there are multiple `rx` or `tx` streams memory can be allocated to each stream:

```
mem=X:Y:X:Y:X:Y
```

Buffer_size (page 27) and rx and tx Streams (page 29) are related to `mem`.

Example

You can split 128MB of memory evenly between the `tx` and `rx` streams using:

```
dagconfig -d0 mem=64:64
```

**Note:** You can not change the stream memory allocations while packet capture or transmission is in progress.

### overlap/nooverlap

Configures the `rx` and `tx` memory hole to be overlapped. This enables in-line forwarding without copying the data across the memory holes.

Example

```
dagconfig overlap
dagconfig nooverlap
```

**Note:** This option is only applicable on firmware images containing TX.

### PCI

Describes the following information about the DAG card:

- The type of PCI used by the DAG card (PCI, PCIx or PCIe)
- Bus speed
- Bus width

Example

```
pcix  133MHz  64-bit
pci  33MHz  32-bit
pcie 8 Gbs 4Lane
```

### rx and tx Streams

Indicates the number of `rx` and `tx` streams are available on the DAG card. Not configurable.

Stream information relates to the setting of mem (page 28).

### rxonly

Configures the memory hole to only receive.

Example

```
dagconfig rxonly
```

### rxpkts/norxpkts

Sets whether this DAG card receives packets.

Example

```
dagconfig rxpkts
dagconfig norxpkts
```

### rxterm/txterm

Sets the rx and tx cable termination to a specified type e.g 100Ω

**Note:** When you set the termination type it applies to both rx and tx cables.

Example

```
dagconfig term75
dagconfig term100
dagconfig term120
dagconfig termext (external termination)
```

### rxtx

Enables both transmit and receive, and splits the memory hole for `rx` and tx.

This allocates 16MB of memory to each transmit stream, and divides the remaining memory between the receive streams.

Example

```
dagconfig rxtx
```

**Note:** This option is only applicable on firmware images containing TX.

### Short

Any packet received shorter than xxx will be flagged as an error. The packet is still received by the host.

If `noerror` enabled...

Example

```
dagconfig short=xxx
```

### slen

Before you begin to capture data you can set the size that you want the captured packets to be. You can do this using the `dagconfig` tool to define the packet snaplength (`slen`).

**Note:** The snaplength value must be a multiple of 8 and in the range 48 to 2040 per card inclusive.

By default, `slen` which is the portion of the packet that you want to capture is set to 48 per card. This means that only the first 48 bytes of each packet will be captured.

If for example you want to capture only the IP header of each packet you may want to set the length to a different value. Alternatively if you want to ensure you capture the whole packet you can set the length to a larger value.

Example

Setting up a DAG 3.7T card with a snap length of 200 bytes:

```
dagconfig -d0 slen=200
```

**Note:** The ERF header is not included in the `slen` value. Therefore a `slen` of 48 will produce a 64-byte capture record made up of 48 bytes plus the number of bytes in the ERF header.

However because the Ethernet record headers occupy 18 bytes instead of the standard 16 bytes, any payload captured will always be 2 bytes less than the `slen` value i.e. a `slen` of 48 will produce a 64-byte record made up of 18 bytes of header and 46 bytes of payload.
For more information on Ethernet records, see Data Formats (page 77) later in this user guide.

### t1_esf / t1_sf / t1_unframed

Sets the T1 line characteristics for the ports in the range A to H or I to P.

### terf_strip16/terf_strip32/noterf_strip

Strips the CRC value (16 or 32 bits) from the packet or sends packet "as is" (`noterf_strip`). The TERF line in the current configuration indicates the current Terf option.

**Note:** Only displayed if the DAG card supports transmit (i.e. has a terf image).

Example

```
dagconfig terf_strip16
dagconfig terf_strip32
dagconfig noterf_strip
```

### txonly

Configures the memory hole to only transmit.

**Note:** Only displayed if the DAG card supports transmit (i.e. has a terf image).

Example

```
dagconfig txonly
```

**Note:** This option is only applicable on firmware images containing TX.

### Txpkts / Notxpkts

Sets whether this DAG card transmits `tx` packets.

Example

```
dagconfig txpkts
dagconfig notxpkts
```

### varlen/novarlen

The DAG 3.7T card is able to capture packets in two ways. They are:

- Variable length capture (`varlen`)
- Fixed length capture (`novarlen`) - (not support on some firmware images)

In **variable length** (`varlen`) mode, the DAG card will capture the whole packet, providing its size is less than the `slen` value. Therefore to use this capture mode effectively you should set the `slen` value to the largest number of bytes that a captured packet is likely to contain. For more information on snaplength, see slen (page 30).

Any packet that is larger than the `slen` value will be truncated to that size. Any packet that is smaller than the `slen` value will be captured at its actual size therefore producing a shorter record which saves bandwidth and storage space.

Example

The example below shows a configuration for variable length full packet capture:

```
dagconfig -d0 varlen
```

In **fixed length** (`novarlen`) mode the card will capture all packets at the same length. Any packet that is longer than the `slen` value will be truncated to that size, in the same way as for `varlen` capture. However any packet that is shorter than the `slen` value will be captured at its full size and then padded out to the size of the `slen` value.

This means that in `novarlen` mode you should avoid large `slen` values because short packets arriving will produce records with a large amount of padding which wastes bandwidth and storage space.

**Note:** Using the `novarlen` option on DAG cards with an on-board Co-Processor (accelerated cards) is not recommended. It may cause excessive loss of packets.

Example

The example below shows a configuration for fixed length packet capture that will produce a 64-byte record:

```
dagconfig -d0 align64 novarlen slen=40
```

**Note:** For Ethernet records a 64 byte record is made up of 46 bytes of payload and 18 bytes of ERF header. For more information on Ethernet records, see Data Formats (page 77) later in this user guide.

### Version information

Details the following information about the connected DAG card:

- Firmware image programmed in the FPGA
- The DAG card serial number
- The MAC address(s) of the DAG card's ports (ethernet cards only).

Example

```
Firmware: dag37dpci_erf_pci_v2_8 3s1500fg456 2006/09/28 10:18:01  (user)
Card Serial: 7000004
MAC Address A: 00:0e:a7:00:57:fa
MAC Address B: 00:0e:a7:00:57:fb
```

## dagconfig options

`dagconfig` is a software utility used to configure and display statistics.

By default all commands, unless otherwise defined, run on device 0 (`-d0`). Commands only apply to one DAG card.

The following is a list options available in `dagconfig`. Not all options listed are applicable to all cards.

| Options: | Description |
|---|---|
| `-1,--porta` | Port A only (default all). Multi-port cards only. |
| `-2,--portb` | Port B only (default all). Multi-port cards only. |
| `-3,--portc` | Port C only (default all). Four-port cards only. |
| `-4,--portd` | Port D only (default all). Four-port cards only. |
| `--porte to --portp` | As above, for extra ports on the 3.7T DAG card. |
| `-c,--concfg <conncfg>` | Connection configuration. Used by the DAG 7.1S only. |
| `-C,--counters` | Outputs the counters. Verbosity levels from 0=(basic / default) to 3=(full). |
| `-d,--device <device>` | DAG device to use. Default is d0. |
| `-e,--extended` | Displays the current extended statistics (non boolean and image dependant). Verbosity levels from 0=(basic / default) to 3=(full). Note: Some images may not contain extended statistics. |
| `-G,--getattribute <getattribute>` | Gets individual attributes by attribute name. Use in conjunction with the --porta or --portb options to get individual only multi-port cards. |
| `-h,--help` | Displays the MAN pages. The information displayed is dynamically based on the DAG card and does not work correctly when there is no DAG card in the system. **Note:** There are a few commands that display even though they are not applicable. |
| `-i,--interval <seconds>` | Interval to repeat in seconds. |
| `-m,--hmon` | Outputs the hardware monitor information. |
| `-n,--voltages` | Outputs the DAG card voltage monitor information. |
| `-S,--setattribute <setattribute>` | Sets individual attributes by attribute name. Use in conjunction with the --porta or --portb options to get individual only multi-port cards. |
| `-s,--statistics` | Outputs the statistics for the DAG card. Verbosity levels from 0=(basic / default) to 3=(full). |
| `-T,--tree` | Outputs the supported Configuration and Status attributes and components with the description and name. Using the -v2 verbosity level also outputs all components and attribute codes. Verbosity levels from 0=(basic / default) to 3=(full). |
| `-t,--txstats` | Outputs the transmit statistics for the DAG card. Where applicable. |
| `-u, --ucounters` | Outputs the universal counters for the DAG card. Where applicable. |
| `-v,--verbose <level>` | Sets the verbosity level, from 0 (basic) to 3 (full). |
| `-V,--version` | Display the DAG card version information. |

**Note:** For cards with more than 2 ports you can select the required port using: -(portnumber) or --(portletter).

## dagthree options

`dagthree` has been depreciated from DAG Software release 3.2 onwards. It has been replaced with `dagconfig`. Both are still valid. Endace recommends that new customer use `dagconfig`.

`dagthree` is a software utility used to configure and display statistics.

By default all commands, unless otherwise defined, run on device 0 (`-d0`). Commands only apply to one DAG card.

The following is a list options available in `dagthree`.

| Option | Description |
|---|---|
| `-1,--1544` | Crystal is 1.544MHz not 2.048MHz |
| `-a,--porta` | Port A only (default both). |
| `-b,--portb` | Port B only (default both). |
| `-c,--counters <c1,c2>` | Display counter statistics c1 and c2. |
| `-d,--device <device>` | DAG device to use. |
| `-f,--framer` | Display E1/T1 Framer statistics. |
| `-h,--help, -?` | Displays the help pages. |
| `-i,--interval <seconds>` | Interval to repeat -s or -c in seconds. |
| `-p,--ptest` | Production test output |
| `-s,--stats` | Display SONET/SDH/PHY statistics |
| `-S,--sticky-counters <seconds>` | Accumulate counter statistics for a number of seconds. |
| `-t,--detect` | Detect card configuration |
| `-u,--sonicid` | Display SONIC user device ID. |
| `-v,--verbose` | Increase verbosity. |
| `-V,--version` | Display version information. |

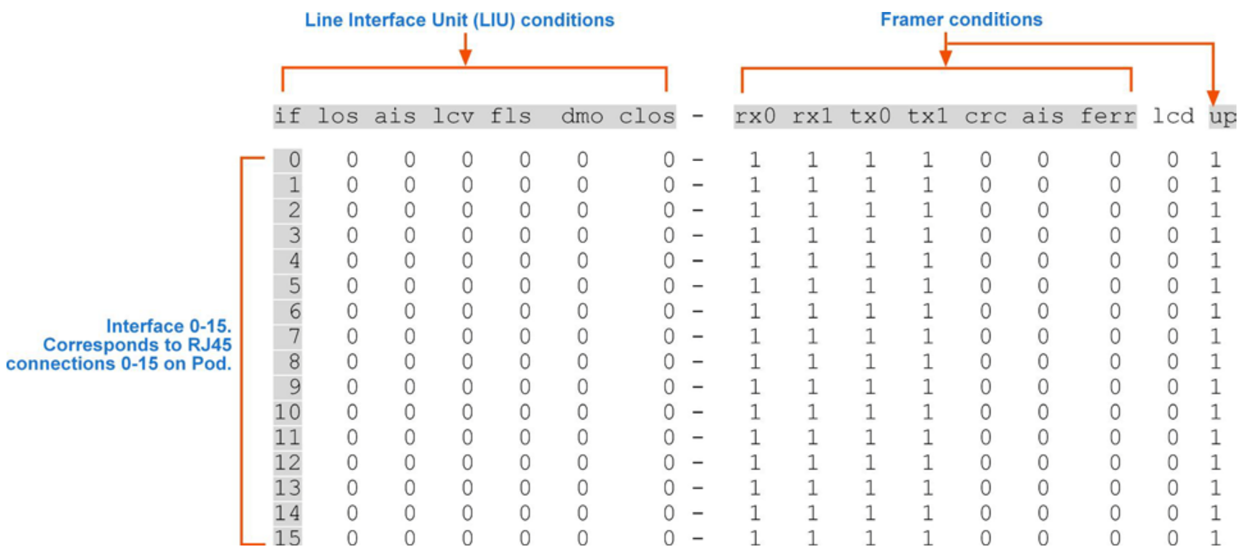
## Viewing the DAG card status

### Interface Status

When you have configured the card for your specific requirements you can view the interface statistics to check the status of each of the links using:

```
dagconfig -d0 -s
```

An example output is shown below:


Line Interface Unit (LIU) conditions
Framer conditions
Interface 0-15. Corresponds to RJ45 connections 0-15 on Pod.

```
if los ais lcv fls  dmo clos - rx0 rx1 tx0 tx1 crc ais  ferr lcd up
 0   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
 1   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
 2   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
 3   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
 4   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
 5   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
 6   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
 7   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
 8   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
 9   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
10   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
11   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
12   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
13   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
14   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
15   0   0   0   0   0    0  -  1   1   1   1   0   0    0    0  1
```

The output shows the condition status of each of the links.

**Note:** "`1`" indicates the condition is present on the link,. "`0`" indicates the condition is not present on the link.

The output includes a number of status bits as they have occurred since the last read. In our example, the read interval is set to 1 sec via the `-i` option.

See the Definitions (page 35) for a full description of each of the status conditions.

### Definitions

A definition of each of the status bits is described below:

| Condition | Definition |
|---|---|
| `if` | Interface or port number (0-15) |
| `los` | LIU Loss of Signal: There is not enough voltage swing on the input line. |
| `ais` | LIU Alarm Indication Signal: The input is receiving only the value "1" |
| `lcv` | LIU Line Code Violation: Data coding is in valid and does not comply with the AMI/HDB3 or B8ZS standard. |
| `fls` | LIU FIFO Limit Status: The data recovery FIFO has overflowed. |
| `dmo` | LIU Drive Monitor Output: Limits the transmit output voltage. |
| `clos` | LIU Cable Loss (dB +/- 1dB): The loss seen on the cable relative to 0dB. |
| `rx0` | The framer has received the value "0". This is useful to ensure the line toggles correctly. |
| `rx1` | The framer has received the value "1". This is useful to ensure the line toggles correctly. |
| `tx0` | The framer has transmitted the value "0". This is useful to ensure the line toggles correctly. |
| `tx1` | The framer has transmitted the value "0". This is useful to ensure the line toggles correctly. |
| `crc` | Framer CRC error: The framer has detected a CRC error. Only valid for E1 CRC. |
| `ais` | Framer Alarm Indication Signal: The framer (internally) is only receiving the value "1" as payload. |
| `ferr` | Framer Error: A framing error has been detected. |
| `lcd` | Loss of Cell Delineation: The ATM demapper experienced a problem locking to the ATM cell stream. |
| `up` | Framer Up: The Framer is locked to the frame sequence. |

### Interpretation

The statistics display which `dagconfig` outputs provides you with detailed information about the status of each of the individual links that you have configured on the card. If any of the conditions on any of the links are not as you expect you may need to review your configuration options and change them accordingly

As a general rule you should use the lowest possible gain control value (eg. short haul) to avoid over amplifying the signal which increases the chance of signal errors.

**For Example:**

For a twisted pair E1 line you could use `Mode 29` (minimum gain control, signal attenuated less < 15dB). However if `dagconfig` reports a poor signal you could increase the amplification by using `Mode 27` (extended long haul/43 dB).

**Caution:**

When using high gain modes ensure the unconnected physical interfaces **are not** configured to capture HDLC data. If configured, the interfaces will receive noise from the adjacent interfaces, amplify it and generate packets with bit errors.

# Configuring HDLC Connections

You can define the HDLC physical channels for the card by using the `dagchan` tool to read a channel configuration file and then create each of the channels defined in that file. `dagchan` is one of the standard tools shipped with your Endace software.

**Note:** To use `dagchan` to configure HDLC channels you must have either the HDLC firmware or the mixed firmware loaded on the card.

You can create a channel configuration file using any common text editor such as Notepad, VI Editor, Emacs etc.

By default `dagchan` deletes all existing channel definitions on the card, and **then** creates the new channel definitions. Use this option if you are configuring a new card or if you want to remove all existing definitions. If the card you are configuring has existing channel definitions that you want to retain, just add new definitions by using the "`-r`" option in the command line.

## Configuration File

You can use the configuration file to do the following:

- Configure HDLC channel connection characteristics.
- Enable/disable RAW Rx.
- Designate channels to either receive or transmit.

**Note:** The type of connections shown in the configuration file reflect the data that is present on the network to which you are connected. Please ensure you understand the type of connection you wish to monitor before beginning to configure the interfaces.

Each line in the configuration file represents a different command. Each command begins with a letter which designates the connection type, followed by a sequence of one or more numbers as shown in the examples later in this chapter. The following letters are valid:

| Command | Description |
|---|---|
| `c` | timeslot connection, |
| `d` | delete connection, |
| `h` | hyper-channel connection, |
| `l` | line connection, |
| `r` | RAW line connection, |
| `s` | sub-channel connection, |
| `lr` | RAW line connection, |
| `hr` | RAW hyper-channel connection, |
| `sr` | RAW sub-channel connection, |
| `cr` | RAW channel connection. |

### Multiple Interfaces

If you wish to configure connections on multiple interfaces you need to repeat the configuration procedure for each interface you want to monitor.

## Receive/Transmit

By default all new connections will receive. To designate specific connections to either receive or transmit you must type "`transmit`" on the line of the file immediately before the configuration line for that connection.

All subsequent connections in the file will also transmit unless you type "`receive`" after which all subsequent connections will receive. This is shown in the sample Hyper-Channel Connection (page 39) `chan.config` file later in this chapter.

**Note:** If you do not type either "`receive`" or "`transmit`" into the configuration file all connections default to receive.

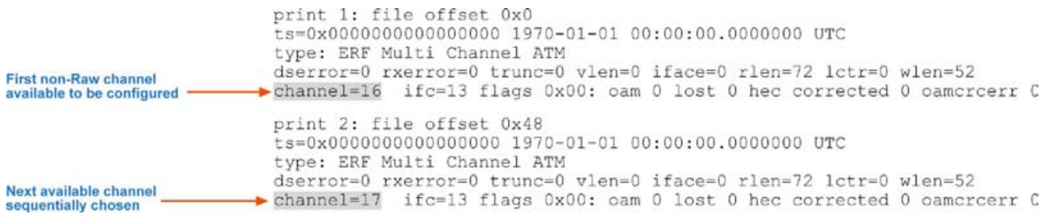
## Transmitting Packets

To transmit packets the number of transmit channels must match the number of channels the packets were captured on. You must set the first channel and the required sequential channels to transmit mode before attempting to transmit. In addition you must configure the transmit channels to match the configuration of the channels on which the packets were captured.

You can determine which channels need to be configured using:

```
dagbits-f traffic_file.erf
```

For example the information printed might be:



```
print 1: file offset 0x0
ts=0x0000000000000000 1970-01-01 00:00:00.0000000 UTC
type: ERF Multi Channel ATM
dserror=0 rxerror=0 trunc=0 vlen=0 iface=0 rlen=72 lctr=0 wlen=52
channel=16  ifc=13 flags 0x00: oam 0 lost 0 hec corrected 0 oamcrcerr C

print 2: file offset 0x48
ts=0x0000000000000000 1970-01-01 00:00:00.0000000 UTC
type: ERF Multi Channel ATM
dserror=0 rxerror=0 trunc=0 vlen=0 iface=0 rlen=72 lctr=0 wlen=52
channel=17  ifc=13 flags 0x00: oam 0 lost 0 hec corrected 0 oamcrcerr C
```

First non-Raw channel available to be configured
Next available channel sequentially chosen

Line five of `print 1: file offset 0x0` and `print 2: file offset 0x48` indicates which channel should be set up. The channel configuration for the above example may be:

```
transmit
h 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
h 0 1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

To implement you configuration file use `dagchan` as follows:

```
dagchan -d0 -c channel_file
```

## Connection ID

### HDLC Connections

The first HDLC connection defined on the board is assigned connection ID 16. Each subsequent connection is assigned a successive ID up to the limit of 496 connections.

A typical HDLC data stream is bit stuffed with a zero bit after every five consecutive one bits to prevent confusion with the HDLC packet delimiter (0x7e or 0111 1110).

### RAW Connections

Raw connections are assigned connection ID 0 through to ID 15. On a RAW connection there is no bit unstuffing.

## Output Record Formats

The output from HDLC connections depends upon the connection type as shown below:

**Note:** See Data Formats (page 77) for detailed descriptions of ERF record formats.

| Connection Type | Output ERF Type | Data Type |
|---|---|---|
| `(-h), (-s), (-c), (-l)` | Type 5 | unbitstuffed unframed |
| `(-r)` | Type 6 | bitstuffed unframed |
| `(-lr), (-hr), (-sr), (-cr)` | Type 8 | bitstuffed framed |

## Connection Types

### Hyper-Channel Connection

A hyper-channel connection is an HDLC connection which occupies one or more timeslots on one interface. This line would look like:

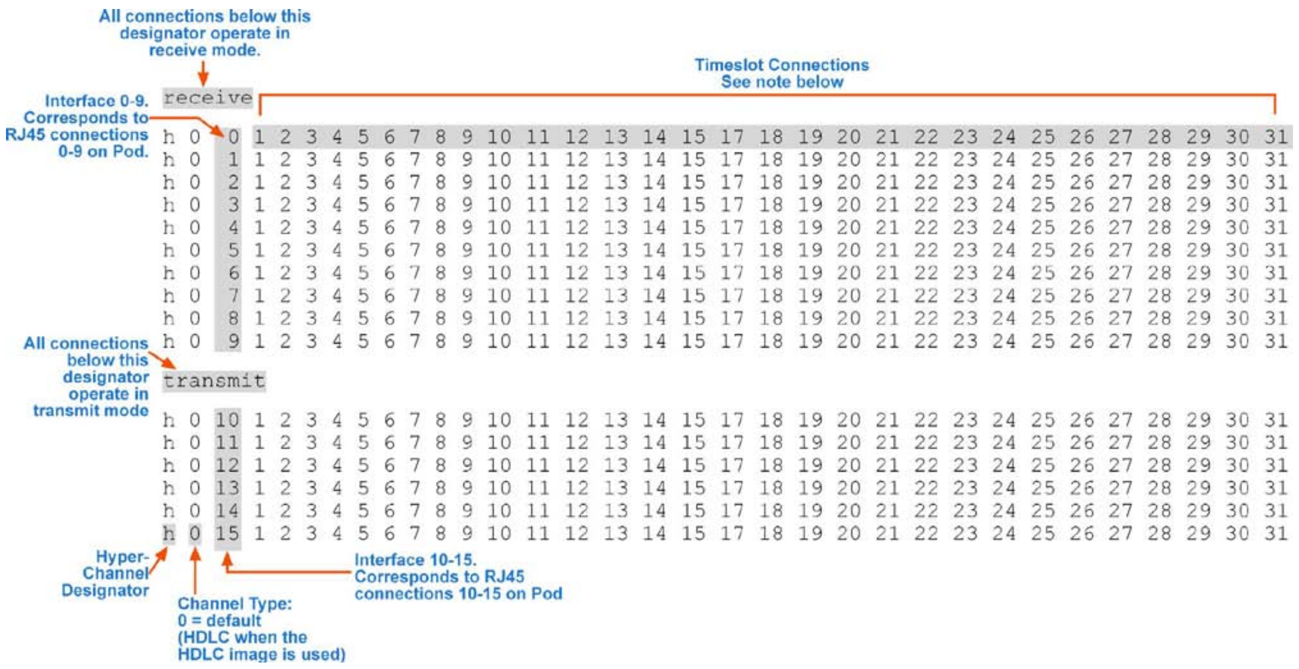
```
h chan_format ifc_num ts_num ts_num ts_num
```

**Note:** you can use up to 32 `ts_num` identifiers

To configure a connection on interface 3, timeslots 0, 1, 2, 26, 27, 28, 29, the line would look like:

```
h  0  3  0  1  2  26  27  28  29
```

An example of a `chan.cfg` file for 16 PCM 30 E1 hyper-channels is shown below:


All connections below this designator operate in receive mode.
Timeslot Connections See note below
Interface 0-9. Corresponds to RJ45 connections 0-9 on Pod.
All connections below this designator operate in transmit mode
Hyper-Channel Designator
Channel Type: 0 = default (HDLC when the HDLC image is used)
Interface 10-15. Corresponds to RJ45 connections 10-15 on Pod

```
receive
h 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 3 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 6 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 7 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 8 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 9 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
transmit
h 0 10 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 11 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 12 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 13 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 14 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
```

**Note:** Timeslots `0` and `16` are not available as PCM 30 uses these for framing and signaling.

### Timeslot Connection

A timeslot connection is a connection which occupies only one timeslot on one interface at 64 kbps. The line would look like:

```
c chan_format ifc_num ts_num
```

To configure a connection on interface 5, timeslot 16, it the line would look like:

```
c   0   5   16
```

## Sub-channel Connection

A sub-channel connection is a connection which occupies one, two, four or seven consecutive bits within a timeslot on one interface.  The line would look like:

```
s chan_format ifc_num ts_num ts_mask
```

To configure a connection on interface 6, timeslot 2, bits 4-7, the line would look like:

```
S   0   6   2   0 0 0 0 1 1 1 1
```

One bit equals 8 kbps. For example, one timeslot at 64k consists of 8x8kbps in a sub-channel.

An 8kbps channel can extends 2 x 8k to 16k, 4 x 8k to 32k, or 7 x 8k to 56k.

**Note:** The DAG 3.7T does not support HDLC transmit on sub-channels.

## Line Connection

A line connection is a connection which occupies all timeslots (i.e. full line rate) on one interface. The line would look like:

```
l chan_format ifc_num
```

To configure a line connection on interface 0 it would look like:

```
1   0   0
```

## RAW Connection

A RAW connection is a connection which occupies all timeslots on one interface and is in RAW mode. RAW mode is when there is no de-framing, or bitunstuffing, only the raw data from the timeslots read by the firmware. The line would look like:

```
r ifc_num
```

To configure a RAW connection on interface 4 it would look like:

```
r 4
```

## Line RAW Connection

A line RAW connection with HDLC firmware is a connection which occupies all timeslots on one interface. The line would look like:

```
lr chan_format ifc_num
```

To configure a raw line connection on interface 0 the line would look like:

```
1r   0   0
```

## Channel RAW Connection

A channel RAW connection with HDLC firmware is a connection which occupies only one timeslot on one interface. The line would look like:

```
cr chan_format ifc_num ts_num
```

To configure a raw connection on interface 5, timeslot 16, the line would look like:

```
cr   0   5   16
```

## Hyper-Channel RAW Connection

A hyper-channel RAW connection is a connection which occupies one or more timeslots on one interface. The line would look like:

```
hr chan_format ifc_num ts_num ts_num ts_num
```

The `bit_offset` field is reserved for future use and must be set to 0.

To configure a raw connection on interface 3, timeslots 0, 1, 2, 26, 27, 28, 29, the line would look like:

```
hr  0  3  0  1  2  27  28  29
```

## Sub-Channel RAW Connection

A sub-channel RAW connection is a connection which occupies one, two, four or seven consecutive bits within a timeslot on one interface.   The line would look like:

```
sr chan_format ifc_num ts_num ts_mask
```

To configure a raw connection on interface 6, timeslot 2, bits 4-7, the line would look like:

```
sr   0   6   2   0 0 0 0 1 1 1 1
```

## Delete Connection

Before you can delete a connection the connection must already exist. When you first start dagchan there are no connections, so all connections that you want to create or delete must exist in the `.cfg` file. The line would look like:

```
d conn_num
```

To delete connection number 17, the line would look like:

```
d 17
```

# Configuring ATM Connections

## Overview

You can define the ATM physical channels for the card by using the `dagchan` tool to read a channel configuration file and then create each of the channels defined in that file. `dagchan` is one of the standard tools shipped with your Endace software and can be found in the `tools` sub-directory.

**Note:** To use `dagchan` to configure ATM channels you must have either the ATM firmware or the mixed firmware loaded on the card.

You can create a channel configuration file using any common text editor such as Notepad, VI Editor, Emacs etc.

By default `dagchan` deletes all existing channel definitions on the card, and **then** creates the new channel definitions. Use this option if you are configuring a new card or if you want to remove all existing definitions. However if the card you are configuring has existing channel definitions that you want to retain you can just add new definitions by using the "`-r`" option in the command line.

## Configuration File

You can use the configuration file to do the following:

- Configure ATM channel connection characteristics,
- Designate channels to either receive or transmit.

**Note:** The type of connections shown in the configuration file reflect the data that is present on the network to which you are connected. Please ensure you understand the type of connection you wish to monitor before beginning to configure the interfaces.

Each line in the configuration file represents a different command. Each command begins with a letter, followed by a sequence of one or more numbers. The following letters are valid line beginnings:

| Command | Description |
|---|---|
| `c` | simple connection, |
| `d` | delete connection, |
| `h` | hyper-channel connection, |
| `l` | line connection, |
| `t` | ATM scrambling on interface, |
| `f` | HEC correction. |

### Multiple Interfaces

If you wish to configure connections on multiple interfaces you need to repeat the configuration procedure for each interface you want to monitor.

## Receive/Transmit

By default all new connections receive. To designate specific connections to transmit you must type “`transmit`” on the line of the file immediately before the configuration line for that connection. All subsequent connections in the file also transmit unless you type “`receive`” after which all subsequent connections receive. This is shown in the sample Hyper-Channel Connection (page 45) `chan.config` file.

**Note:** If you do not type either “`receive`” or “`transmit`” into the configuration file all connections default to receive.

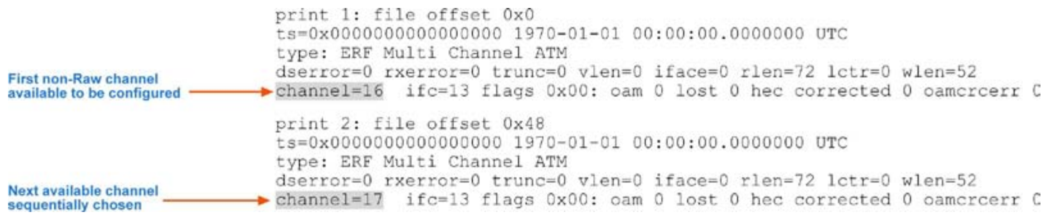
## Transmitting Packets

To transmit packets the number of transmit channels must match the number of channels the packets were captured on. You must set the first channel and the required sequential channels to transmit mode before attempting to transmit. In addition you must configure the transmit channels to match the configuration of the channels on which the packets were captured.

You can determine which channels need to be configured using:

```
dagbits-f traffic_file.erf
```

For example the information printed might be:


First non-Raw channel available to be configured
Next available channel sequentially chosen

```
print 1: file offset 0x0
ts=0x0000000000000000 1970-01-01 00:00:00.0000000 UTC
type: ERF Multi Channel ATM
dserror=0 rxerror=0 trunc=0 vlen=0 iface=0 rlen=72 lctr=0 wlen=52
channel=16  ifc=13 flags 0x00: oam 0 lost 0 hec corrected 0 oamcrcerr C

print 2: file offset 0x48
ts=0x0000000000000000 1970-01-01 00:00:00.0000000 UTC
type: ERF Multi Channel ATM
dserror=0 rxerror=0 trunc=0 vlen=0 iface=0 rlen=72 lctr=0 wlen=52
channel=17  ifc=13 flags 0x00: oam 0 lost 0 hec corrected 0 oamcrcerr C
```

Line five of `print 1: file offset 0x0` and `print 2: file offset 0x48` indicates which channel should be set up. The channel configuration for the above example may be:

```
transmit
h 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
h 0 1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

To implement you configuration file use `dagchan` as follows:

```
dagchan -d0 -c channel_file
```

## Output Record Formats

The output from ATM connections depends upon the connection type as shown below:

See Data Formats (page 77) for further information on ERF record formats.

| Connection Type | Output ERF Type |
|---|---|
| `(-c), (-d), (-h), (-l),(-t), (-f)` | Type 7 |
| `AAL5 Reassembly` | Type 9 |
| `AAL2 Reassembly` | Type 12 |

## Connection Types

### Hyper-Channel Connection

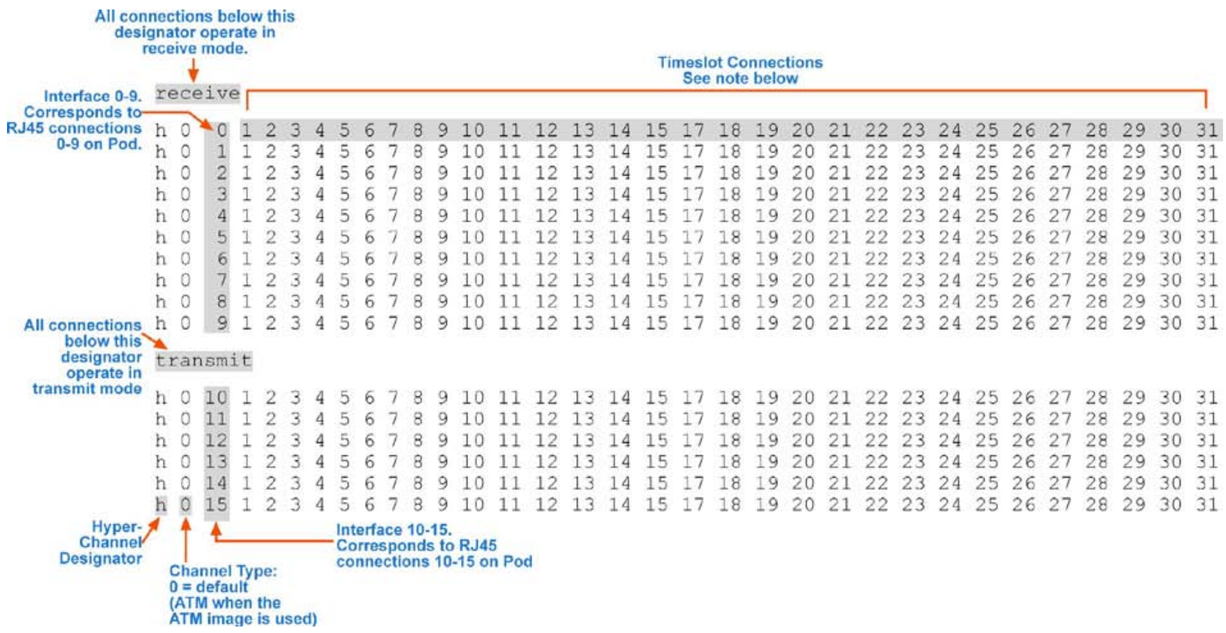
A hyper-channel connection is a connection which occupies one or more timeslots on one interface. The line would look like:

```
h chan_format ifc_num ts_num ts_num ts_num
```

To configure a connection on interface 3, timeslots `0, 1, 2, 26, 27, 28, 29`, the line would look like:

```
h  0  3  0  1  2  26  27  28  29
```

An example of a `chan.cfg` file for 16 PCM 30 E1 hyper-channels is shown next:


All connections below this designator operate in receive mode.
Timeslot Connections See note below
Interface 0-9. Corresponds to RJ45 connections 0-9 on Pod.
All connections below this designator operate in transmit mode
Hyper-Channel Designator
Channel Type: 0 = default (ATM when the ATM image is used)
Interface 10-15. Corresponds to RJ45 connections 10-15 on Pod

```
receive
h 0  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  3 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  6 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  7 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  8 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  9 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
transmit
h 0 10 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 11 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 12 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 13 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 14 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
```

**Note:** Timeslots `0` and `16` are not available as PCM 30 uses these for framing and signaling.

### Timeslot Connection

A timeslot connection is a connection which occupies only one timeslot on one interface. The line would look like:

```
c chan_format ifc_num ts_num
```

The field `chan_format` is reserved for future use and must be set to `0`.

To configure a connection on interface 5, timeslot 16, the line would look like:

```
c   0   5   16
```

### Line Connection

A line connection is a connection which occupies all timeslots (i.e. full line rate) on one interface. The line would look like:

```
l chan_format ifc_num
```

To configure a line connection on interface 0 it would look like:

```
1   0   0
```

## ATM Scrambling on Interface

The DAG 3.7T card is equipped to unscramble any interface using the self synchronizing scrambling polynomial specified in ITU I.432. When an interface is unscrambled, all connections on the interface will be unscrambled.

To enable scrambling on an interface, the line would look like:

```
t ifc_num
```

In order to configure interface 3 to unscramble ATM cells, the line would look like:

```
t   3
```

## HEC Connection on Interface

The DAG 3.7T card can correct single bit errors in the HEC field as detailed in the ITU I.432 specification. When an interface is correcting HECs, all connections on the interface will be corrected.

To enable HEC correction on an interface, the line would look like this:

```
f ifc_num
```

To configure interface 3 to perform HEC correction, the line would look like:

```
f   3
```

## Delete Connection

Before you can delete a connection the connection must already exist. When you first start dagchan there are no connections, so all connections that you want to create or delete must exist in the `.cfg` file. The line would look like:

```
d conn_num
```

To delete connection number 17, the line would look like:

```
d 17
```

# Configuring Mixed ATM and HDLC Connections

## Using Mixed Firmware

The mixed image allows you to capture both ATM and HDLC data on the same DAG card.

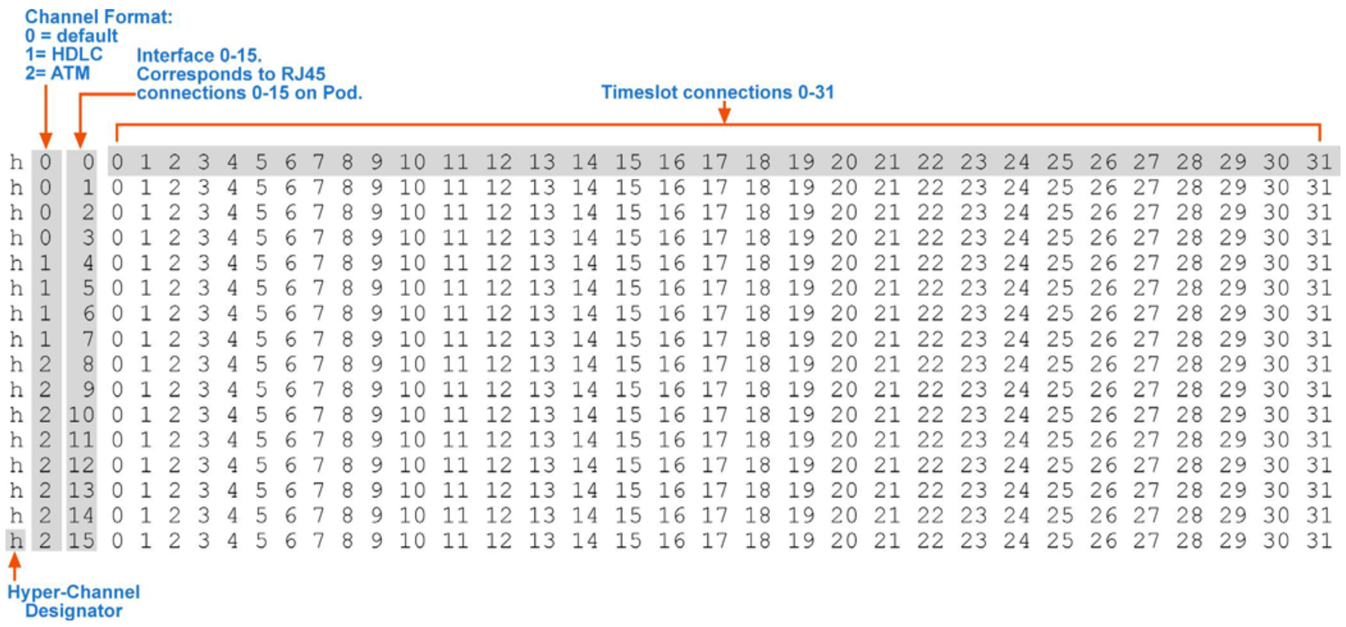
**Note:** The mixed firmware does **not** support transmit. If you wish to transmit data as well as receive you must use **either** the ATM **or** the HDLC firmware image.

You can use `dagchan` to set the `chan_format` for individual connections to either HDLC or ATM. By default the mixed firmware image is set to HDLC.

Valid channel types are :

- `0` = Default
- `1` = HDLC
- `2` = ATM

An example a `chan.config` file for mixed HDLC and ATM Hyper-Channel connections occupying all timeslots is shown below:



```
h 0  0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  2 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 0  3 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 1  4 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 1  5 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 1  6 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 1  7 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 2  8 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 2  9 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 2 10 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 2 11 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 2 12 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 2 13 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 2 14 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
h 2 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
```

**Note:** Because the mixed firmware does not support transmit you do not need to set individual connections to receive or transmit. All connections will automatically default to receive only.

For detailed information on configuring specific HDLC and ATM connections, see Configuring HDLC Connections (page 37) and Configuring ATM Connections (page 43).

# Using your DAG card to capture data

## Introduction

This chapter describes how to complete the following operations for a DAG card:

## Basic data capture

`dagsnap` is a software utility used to write to disk the raw data captured from a DAG card.

Data is collected in Extensible Record Format (ERF) which can then be viewed using `dagbits`, or converted to other formats using `dagconvert`.

When capturing high speed data Endace recommends you use `dagsnap`, see Capturing data at high speed (page 51).

For further information on the software utilities see:

- dagsnap (page 50)
- dagbits (page 52)
- dagconvert (page 55)

### Starting a capture session

To start the capture session type the following at the prompt:

```
dagsnap -d0 –v -o tracefile
```

(where "0" is the device number of the DAG card you wish to capture data from)

**Note:** You can use the `-v` option to provide user information during a capture session although you may want to omit it for automated trace runs.

By default, `dagsnap` runs indefinitely. To stop, use *CTRL+C*. You can also configure `dagsnap` to run for a fixed time period then exit.

## dagsnap

`dagsnap` is a data capture software utility.

The following is a list of the options available in `dagsnap`.

| Option | Description |
|---|---|
| `-d DEVICE`<br>`--device DEVICE` | Use the DAG device referred to by DEVICE. Supported syntax includes 0, dag1, and /dev/dag3 to refer to DAG cards, and 0:2, dag1:0, and /dev/dag2:0 to refer to specific streams on cards. |
| `-h, -?`<br>`--help, --usage` | Display usage (help) information. |
| `-j` | Maximize disk writing performance by only writing data to disk in chunks. This option may not be available on all operating systems. |
| `-m NUM` | Write at most NUM megabytes of data per call to the DAG API (default is 4 MiB). |
| `-o FILE`<br>`--fname FILE` | Write the captured packets to FILE, in ERF format (default is standard output). |
| `-s NUM`<br>`--runtime NUM` | Run for NUM seconds, then exit. |
| `-v`<br>`--verbose` | Increase output verbosity. When `dagsnap` is run with this command three columns of data are reported every second. These columns contain<br>1. The cumulative total of data written out from the DAG card.<br>2. The buffer occupancy. Small values indicate no packet loss.<br>3. The rate at which data is currently being written. |
| `-V, --version` | Display version information. |
| `-w,--wait SEC` | Delay(wait) in seconds before capture and after. |

## Capturing data at high speed

As the DAG 3.7T card captures packets from the network link, it writes a record for each packet into a large buffer in the host computer's main memory.

To avoid packet loss, the user application reading the record, such as `dagsnap`, must be able to read records out of the buffer as fast or faster than they arrive. If not the buffer will eventually fill and packet records will be lost.

If the user process is writing records to hard disk, it may be necessary to use a faster disk or disk array. If records are being processed in real-time, a faster host CPU may be required.

In Linux and Free BSD, when the computer buffer fills, the following message displays on the computer screen:

```
kernel: dagN: pbm safety net reached 0xNNNNNNNN
```

The same message is also printed to log `/var/log/messages` file. In addition, when the computer buffer fills the "Data Capture" LED on the card will flash or flicker, or may go OFF completely.

In Windows no screen message displays to indicate when the buffer is full. Please contact Endace Customer Support at support@endace.com for further information on detecting buffer overflow and packet loss in Windows.

### Detecting Packet Losses

Once the buffer fills, any new packets arriving will be discarded by the DAG 3.7T card until some data is read out of the buffer to create free space.

You can detect any such losses by observing the Loss Counter (`lctr` field) of the Extensible Record Format (ERF). See Data Formats (page 77) later in this User Guide for more information on the Endace ERF record format.

### Increasing Buffer Size

You can increase the size of the host computer buffer to enable it to cope with bursts of high traffic load on the network link.

For information on increasing the buffer size, see buffer_size (page 27).

## Viewing captured data

Data captured in ERF format can be viewed with `dagbits`. For further details on how to use `dagbits`, see dagbits (page 52).

**Note:** `dagbits` decodes and displays ERF header fields and packet contents are displayed as a Hex dump only. To decode higher level protocols, Endace recommends using a third party application, see Using third party applications (page 56).

Examples

Test live traffic on dag0, stream 0 for 60 seconds running the lctr, flags and fcs tests:

```
dagbits -vvc -d0:0 -s60 lctr flags fcs
```

To read a trace log file using dagbits:

```
dagbits -vvc print -f logname.log | less
```

To check for errors in the trace:

```
dagbits -vvc ltcr flags fcs -f logname.log
```

If `dagbits` reaches the end of the traffic and prints its report then the ERF records were valid.

### dagbits

`dagbits` is a software utility used to view and test ERF records. `dagbits` can receive data from either:

- directly from the DAG card (using the `-d` option), or
- a ERF data file created by `dagsnap`.

The following is a list options available in `dagbits`:

| Options | Description |
|---|---|
| `-0`<br>`-16`<br>`-32` | ERF records contain no Link-Layer CRCs.<br>ERF records contain 16 bit Link-Layer CRCs (PoS).<br>ERF records contain 32 bit Link-Layer CRCs (PoS and Ethernet). |
| `-a` | Set legacy format to ATM (this is the default). |
| `-b` | Treat ERF timestamps as big-endian. |
| `-c` | Print real-time progress reports as `dagbits` captures traffic. This is a useful indicator that a test is running correctly. |
| `-C NUM` | Sets CRC correction factor for BTX test (0, 16 or 32 bits). |
| `-d DEVICE`<br>`--device DEVICE` | Use the DAG device referred to by DEVICE. Supported syntax includes 0, dag1, and /dev/dag3 to refer to DAG cards, and 0:2, dag1:1, and /dev/dag2:0 to refer to specific streams on cards. |
| `-D NUM` | Introduces a NUM nanosecond delay between processing each record. |
| `-e` | Set legacy format to Ethernet (default: ATM). |
| `-E NUM` | Halt operation after a maximum of NUM errors. This option prevents `dagbits` from creating extremely large output files when being redirected to a file. |
| `-f FILE` | Read captured data from FILE. |
| `-h, -?`<br>`--help, --usage` | Display usage (help) information. |
| `-i API` | Use "API" interface for live DAG API capture. Possible options are:<br>0 DAG 2.4 legacy API interface [dag_offset(3)].<br>1 DAG 2.5 API interface [dag_advance_stream(3)].<br>2 DAG 2.5 API interface [dag_rx_stream_next_record(3)]. |
| `-I` | Assume the ERF contains color information in the pad and offset bytes (for Ethernet ERFs) or HDLC header bytes (for PoS ERFs) and display this information as a packet classification and destination memory buffer. |
| `-j NUM` | Set the threshold for the jitter test to NUM microseconds. |
| `-m NUM` | Print the first NUM errored records only, and then continue to count errors silently for the duration of the session. |

| | |
|---|---|
| `-n NUM` | Expected number of packets to receive. Returns an error if the actual number is different. |
| `-p` | Set legacy format to PoS (default: ATM). |
| `-P PARAMS` | DAG 3.5S capture parameters. |
| `-q` | Quiet. This instructs `dagbits` to suppress summary information when terminating. Error messages are not affected by this option. |
| `-r NUM` | Set legacy format record lengths to NUM. |
| `-R NUM` | When used in conjunction with the rlen test, indicates the RLEN of ERF records to match against. NUM. |
| `-s` | Check for strictly monotonic (increasing) timestamps, rather than monotonic (non-decreasing). Affects the behavior of the mono test. With strict checking it is an error for consecutive timestamps to be equal; they must always increase. |
| `-S NUM` | Terminate `dagbits` after NUM seconds of capture. This option only makes sense when capturing packets from a DAG card (i.e. when used in conjunction with the -d flag). |
| `-t NUM` | Terminate `dagbits` if any ERF record type does not match NUM. |
| `-U NUM` | Process at most NUM records in one pass. This option enables the user to reduce the performance of `dagbits` for various purposes. See also -D. |
| `-v`<br>`-vv`<br>`--verbose` | Increase verbosity of `dagbits`. This option increase the amount of data displayed when printing an ERF record due to the print test or errors in other tests. -v will print payload contents, -vv will print payload contents and an accompanying ASCII dump of the packet payload. |
| `-V, --version` | Display version information. |
| `-w` | Instruct dagbits to treat all warnings as errors. |
| `-W NUM` | When used in conjunction with the wlen test, the wire length of ERF records must be exactly NUM bytes. |
| `-z` | Stop when no traffic is received for one second. |

`dagbits` takes several options that serve as parameters to particular tests. Available tests include monotonic time-stamp increment and frame checksum (FCS, aka CRC) validation. See the `dagbits` help for further details.

## Converting captured data

`dagconvert` is the software utility that converts captured data from ERF format to Pcap (and other formats). Once in non ERF format the data can be read using third party applications (page 56).

`dagconvert` can also be used to capture data directly into pcap format.

### Examples

To read from DAG card 0 and save to a file in ERF format:

```
dagconvert -d0 -o outfile.erf
```

To read from DAG card 0 and save to a file in pcap format:

```
dagconvert -d0 -T dag:pcap -o outfile.pcap
```

To convert a file from ERF format to pcap format:

```
dagconvert -T erf:pcap -i infile.erf -o outfile.pcap
```

To convert a file from pcap format to ERF format, ensuring the ERF records are 64-bit aligned (and therefore suitable for transmission using dagflood):

```
dagconvert -T pcap:erf -A 8 -i infile.pcap -o outfile64.erf
```

To capture from DAG Card 0 using a BPF filter:

```
dagconvert -d0 -o outfile.erf -b "host 192.168.0.1 and tcp port 80"
```

To capture from DAG card 0 using ERF filtering:

```
dagconvert -d0 -o outfile.erf -f "rx,a"
```

To capture from DAG card 0 to a series of files of size 128 MB:

```
dagconvert -d0 -o outfile.erf -r 128m
```

The first file created is labeled `outfile0000.erf`, once the file size reaches 128MB, a second file is created. The second is labeled `outfile0001.erf` etc.

ATM traffic filtering (ATM, AAL5, MC_AAL5 record types) by VPI and VCI is also possible by using the SUNATM DLT, which includes VPI/VCI information. The following example shows how to use `dagconvert` to achieve this:

```
dagconvert -v -i atm.erf -o atm_filtered.erf -y SUNATM -b "vpi 10 and vci 12"
```

## Dagconvert

`dagconvert` is a software utility for converting data to various file formats. Supported formats are:

| File format | Description |
|---|---|
| dag | Read ERF records directly from DAG card (input only). |
| erf | ERF (Extensible Record Format) file (input and output). |
| atm | Legacy ATM files (input only). |
| eth | Legacy Ethernet files (input only). |
| pos | Legacy PoS files (input only). |
| null | Produces no input or output. |
| pcap | pcap(3) format file (input or output). |
| prt | ASCII text packet dump (output only). |

Data can be input from a file or captured from a DAG card. `dagconvert` can be used for converting data captured from a DAG card to pcap format. This allows the trace file to be used with tools that support the pcap file format. Also the reverse is possible, where data can be converted to ERF format for use in other dag utilities. The following is a list of options available in `dagconvert`.

| Options | Description |
|---|---|
| `-A NUM` | Set the record alignment of the ERF to NUM bytes (ERF only). |
| `-b EXPRESSION` | Specify a tcpdump(1) style BPF expression to be applied to the packets. |
| `-c 0\|16\|32` | Specify the size (in bits) of the frame checksum (FCS) (pcap(3) only). |
| `-d DEVICE`<br>`--device DEVICE` | Use the DAG device referred to by DEVICE. Supported syntax includes 0, dag1, and /dev/dag3 to refer to DAG cards, and 0:2, dag1:1, and /dev/dag2:0 to refer to specific streams on cards. |
| `-f FILTERS` | A comma-delimited list of filters to be applied to the data. Supported filters are:<br>• `rx` Filter out rx errors (link layer).<br>• `ds` Filter out ds errors (framing).<br>• `trunc` Filter out truncated packets.<br>• `a,b,c,d` Filter on indicated port/interface(s). |
| `-F` | Select fixed length output (ERF only). |
| `-G NUM` | Set the GMT offset to NUM seconds (pcap(3) only). |
| `-h, -?, --help, --usage` | Display usage (help) information. |
| `-i FILES` | Name(s) of the input file(s). If more than one filename is given, the ERF records from the files will be merged in timestamp order to the output. |
| `-o FILE` | Name of the output file. |
| `-r NUM` | Rotate the output file after NUM bytes. Add `k` (kilobytes), `m` (megabytes), `g` (gigabytes) and `t` (terabytes) suffixes. |
| `-s NUM` | Set the snap length to NUM bytes. |
| `-t NUM` | Capture from the DAG card for NUM seconds. |
| `-T atm\|dag\|erf\|eth\|pcap\|pos : erf\|pcap\|prt` | Input and output types. See the DESCRIPTION section above for more information about the input and output types. |
| `-v, --verbose` | Increase output verbosity. |
| `-V, --version` | Select variable length output (ERF only). Display version information. |
| `-y <DLT>` | This sets the pcap data link type to be used for BPF filtering (-b) and for pcap output. Previously only one DLT was mapped to each ERF type.<br>Supported DLT types (case insensitive):<br>EN10MB: Ethernet    DOCSIS : Ethernet<br>CHDLC : HDLC    PPP_SERIAL : HDLC<br>MTP2 : HDLC    ATM_RFC1483 : ATM, AAL5<br>SUNATM : ATM, AAL5 |

**Note:** Not all options are applicable to all DAG cards.

## Using third party applications

Once the captured data is in Pcap format you can use third party applications to examine and process the data. The third party applications include:

- Wireshark /Tshark (formerly Ethereal /Tethereal)
- TCPDump
- Libpcap
- SNORT
- Winpcap, etc.

**Note:** Wireshark can also read ERF formatted data.

## Transmitting captured data

### Configuration

The DAG 3.7T card is able to transmit as well as receive packets and can capture received traffic **while** transmitting. This allows you to use capture tools such as `dagsnap`, `dagconvert`, and `dagbits` while `dagflood` is sending packets.

To configure the DAG 3.7T card for transmission, you must allocate some memory to a transmit stream. By default, 16 MB of memory is allocated to the tx stream and the remainder is allocated to the rx stream. For information on setting the Memory allocation see mem (page 28).

**Note:** You can not change the stream memory allocations while packet capture or transmission is in progress.

### Explicit Packet Transmission

The operating system does not recognize the DAG 3.7T card as a network interface and will not respond to ARP, ping, or router discovery protocols.

The DAG 3.7T card will only transmit packets that are explicitly provided by the user. This allows you to use the DAG 3.7T card as a simple traffic load generator.

You can also use it to retransmit previously recorded packet traces. The packet trace is transmitted as fast as possible. The packet timing of the original trace file is not reproduced.

### Trace Files

You can use `dagflood` to transmit ERF format trace files, providing the files contain **only** ERF records of the type matching the current link configuration.

When you use DAG cards with multiple ports, ensure all ports referred to by the Trace file are active. This ensures the `dagflood` traffic is not blocked when trying to delivering data to an inactive port. Check the interface status output for the DAG card and ensure the link status for all required destination ports is active. See Viewing the DAG card statistics. (page 34)

For further information on using `dagflood` please refer to the *EDM04-03 dagflood User Manual* available from Endace Customer Support at https://www.endace.com/support.

In addition the length of the ERF records to be transmitted must be a multiple of 64-bits. You can configure this when capturing packets for later transmission by setting 64-bit alignment using the `dagconfig align64` command.

If packets have been captured without using the `align64` option you can convert the trace files so that they can be transmitted by using dagconvert (page 55) as shown below:

```
dagconvert -v -i tracefile.erf -o tracefile.erf -A8
```

Alternatively if you are unsure if a trace file is 64-bit aligned you can test the file using dagbits (page 52) as shown below:

```
dagbits -v align64 -f tracefile.erf
```

If you do not have any ERF trace files available, you can use `daggen` to generate trace files containing simple traffic patterns. This allows the DAG 3.7T card to be used as a test traffic generator.

For further information on using `daggen` please refer to the *EDM04-06 Daggen User Guide* available from Endace Customer Support at https://www.endace.com/support.

# Configuring Extended Functions

## Overview

The embedded XScale processor provides the means to perform the following extended functions:

- Reassembly of AAL2/AAL5 frames,
- Inverse Multiplexing (IMA) monitoring over ATM,
- Frame filtering over HDLC.

**Note:** You can run IMA and HDLC filtering simultaneously however this "mixed" feature **does not** provide transmit functionality.

Before you can make use of these extended functions you must ensure you have completed the following steps:

- Load the ATM, HDLC or mixed firmware image depending upon the function you want to perform,
- Set the correct physical layer characteristics required to receive data,
- Configure the channels for the type of data you want to capture,
- Check (using `dagsnap`) that you are able to capture ATM or HDLC traffic or both as appropriate.

See the following chapters for details for the above steps:

- Configuring the DAG card (page 23)
- Configuring HDLC Connections (page 37)
- Configuring ATM Connections (page 43)

### Loading the Images

To make use of the extended functionality provided by the XScale, you must load the appropriate XScale images. You can do this using the `dagrom`. tool and the region (`-c`) switch. There are three images required. They are:

```
- Bootloader,
- Kernel, and
- Filesystem.
```

- Load the bootloader image using:

```
dagrom -rvl -cb -f d37t_bootloader.rom
```

- Then load the kernel using:

```
dagrom -rvl -ck -f d37t_kernel.rom
```

- Then load the file system using:

```
dagrom -rvl -cf -f d37t_combined.ext2.gz
```

### Starting the XScale

You can start the XScale using one of the following "demo" tools with the "`-x`" option depending upon the function you want to perform:

- `dagaal5demo` for AAL2/AAL5 reassembly
- `dagimademo` for IMA
- `daghdlcdemo` for HDLC filtering

**Note:** You can also make a connection to the embedded processor by using the `dagema` library. For more information on the `dagema` library please contact Endace Support at support@endace.com.

### Directing Data to the XScale

In normal circumstances when you run `dagconfig`, data is directed from the line directly to the host computer.

However to enable any of the extended functions the data must be directed

- from the line, **then**
- to the XScale processor, **then**
- to the host computer.

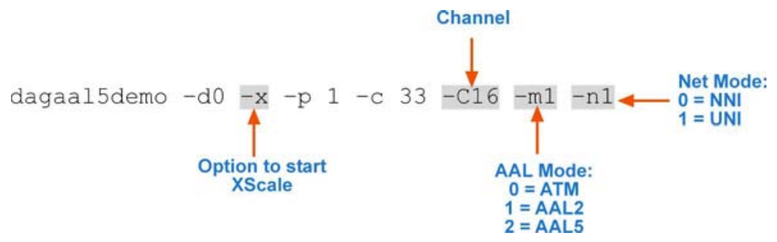
All the "demo" tools listed in the section above do this by default. Alternatively you can use the `dag_set_mux()` function from the DAG 3.7T library.

## Using the AAL Reassembler

The AAL Reassembler allows you to reassemble AAL5 or AAL2 frames on the DAG 3.7T card without involving the host computer in processing. The reassembler receives ATM traffic from the line and then sends it to the host either unchanged, dropped or reassembled into AAL5 or AAL2 frames depending upon the configuration used.

You can use different configurations on different virtual connections, allowing only the required data to be reassembled and other data to be either preserved or rejected.

- To configure an ATM connection to reassemble either AAL2 or AAL5 frames use `dagaal5demo` ensuring you use the `-x` option to start the XScale on the **first** connection:



```
dagaal5demo -d0 -x -p 1 -c 33 -C16 -m1 -n1
```

- Configure as many additional connections to reassemble AAL frames as required. You do not need to use the –x option on additional connections.
- Start capturing the data using:

```
dagsnap -d0 -v -o output_filename
```

For more information on the specific configuration options available for the AAL2/AAL5 reassembler, please refer to the *EDM04-13 SAR API Programming Guide* available from Endace Support at support@endace.com.

## Using IMA

### Overview

The Inverse Multiplexing ATM (IMA) functionality supports both IMA Monitor Mode and IMA Transmit. Operationally there is no distinction between monitor and transmit except that in the case of monitor mode the group is configured with receive links only. In transmit mode the group is configured with both receive and transmit links.

For more information on the specific configuration options available for the IMA Monitor, please refer to the *EDM04-18 IMA Host API Programming Guide* available from Endace Support at support@endace.com.

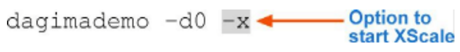
### IMA Monitor

**Note:** The IMA Monitor is intended for use in wire-tap or monitoring mode. You can not use it to terminate an IMA connection.

The IMA Monitor implements a subset of the IMA standard that allows you to monitor/tap an IMA connection and receive fully de-multiplexed ATM cells, as if it were the receiver.

By default the IMA Monitor blocks ATM cells on all interfaces unless you configure a group specifically or use auto-configuration. However running `dagimademo` sets the monitor to auto-configuration mode. In this mode the monitor will auto-configure any groups that it detects on the line.

- To run the IMA Monitor in auto-configuration mode on all interfaces  use:



```
dagimademo -d0 -x
```

- If the IMA monitor detected groups on one or more of the interfaces you will be able to start capturing the re-ordered cell data using:

```
dagsnap -d0 -v -o output_filename
```

## IMA Transmit

**Note:** IMA transmit only supports symmetrical configuration

IMA transmit implements a subset of the IMA standard that allows you to terminate IMA connections.When setting up IMA transmit, which requires both transmit and receive links, you must configure the firmware to recognize the ERF header direction bit. This allows packets to be directed to both the line and the host.

- To configure the firmware to recognize the ERF header direction bit use:

```
dagbug -d0 -v -c “iow 400 2”
dagbug -d0 -v -c “iow 404 2”
dagbug -d0 -v -c “iow 408 0x80”
```

- Alternatively you can use an equivalent DAG 3.7T API function call to:

```
dag_set_mux
```

The file `dagema_stub.c` which can be found in the `tools/embedded` directory contains an number of example of group configurations. You can run `dagema_stub` to create example groups on two DAG 3.7T cards. Please refer to the *EDM04-18 IMA Host API Programming Guide* available from Endace Support at support@endace.com for definitions of the API functions.

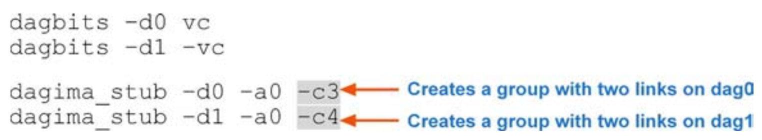
**Note:** Before configuring a group you must ensure you are receiving cells from the card. You can do this by running `dagbits` for example. If you are not receiving cells, cells destined for the host can become backed up in the DAG cards transmit path.

### For Example

In this example there are two DAG cards configured as `dag0` and `dag1` and connected by at least two physical links.

On `dag0` the **transmit** channels have channel ID `16 to 31` and the **receive** channels have channel ID `32 to 47`. On `dag1` these are reversed with the **receive** channels having channel ID `16 to 31` and the **transmit** channels have channel ID `32 to 47`.

To configure a group you can use:



```
dagbits -d0 vc
dagbits -d1 -vc

dagima_stub -d0 -a0 -c3
dagima_stub -d1 -a0 -c4
```

Creates a group with two links on dag0
Creates a group with two links on dag1

There are a number of options available in `dagima_stub` which you can use to query the state of the group and it links.

To transmit ATM cells using IMA you must set the ERF headers IMA Group handle to correspond to the group through which you want to transmit. You can use the program `imafllood` located in tools/embedded which contains an example of how to transmit cells to the intended group.

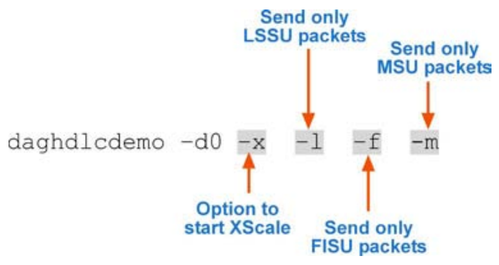
`imaflood` queries the group to determine how many cells it is expecting. This is important as the requested number of cells are then transmitted. If you do not query the group first, the host will flood the card with cells which will prevent IMA from receiving cells from the line.

## Using the HDLC Filter

The HDLC filter is designed to allow Layer 2 filtering and filtering on HDLC packets on the DAG 3.7T card without involving the host computer in processing. The filter receives HDLC traffic from the line and then either sends it to the host unchanged or dropped, depending upon the filter configuration used.

By default the filter drops the packets that match the filter and sends all other packets to the host. If you do not setup any filters all packets will be sent to the host. You can set up filters using one or more of the `-l`, `-f` or `-m` options.

- To run the HDLC Filter on all interfaces with filters setup use:



```
daghdlcdemo -d0 -x -l -f -m
```

- Start capturing the filtered data using:

```
dagsnap -d0 -v -o output_filename
```

For more information on the specific configuration options available for the HDLC Filter, please refer to the *EDM04-12 DAG 3.7T HDLC Filtering Guide* available from Endace Support at support@endace.com.

## Using HDLC and IMA Together

You can use IMA and the HDLC Filter at the same time by running `dagimademo` and `daghdlcdemo` consecutively. This will allow you to capture both HDLC and ATM traffic on the one DAG Card

**Note:** Ensure you do not use the `-x` option with the second `demo` tool that you run. If you do you will restart the XScale and overwrite the configuration options set by the first `demo` tool.

# Synchronizing Clock Time

## Overview

DAG cards have sophisticated time synchronization capabilities. This allows for high quality timestamps and optional synchronization to an external time standard.

The core of the DAG synchronization capability is known as the DAG Universal Clock Kit (DUCK).

A clock in each DAG card runs independently from the computer clock. The DAG card's clock is initialized using the computer clock, and then free-runs using a crystal oscillator.

Each DAG card's clock can vary relative to a computer clock, or other DAG cards.

## DUCK Configuration

The DUCK (DAG Universal Clock Kit ) is designed to reduce time variance between sets of DAG cards or between DAG cards and coordinated universal time [UTC].

You can obtain an accurate time reference by connecting an external clock to the DAG card using the time synchronization connector. Alternatively you can use the host computer's clock in software as a reference source without any additional hardware.

Each DAG card can also output a clock signal for use by other DAG cards.

## Common Synchronization

The DAG card time synchronization connector supports a Pulse-Per-Second (PPS) input signal, using RS-422 signaling levels.

Common synchronization sources include GPS or CDMA (cellular telephone) time receivers.

Endace also provides the TDS 2 Time Distribution Server modules and TDS 6 expansion units that enable you to connect multiple DAG cards to a single GPS or CDMA unit.

For more information, please refer to the Endace website at https://www.endace.com/support, or the *EDM05-01 Time Distribution Server User Guide*.

## Network Time Protocol

NTP (Network Time Protocol) can be used to synchronize a computer clock to a network based reference. When the NTP daemon starts, it exchanges packets with network time servers to establish the correct time. If the computer clock is significantly different, the NTP can adjust the computer clock in a single large 'step'. Over time, NTP adjusts the rate of computer clock to minimize the offset from its reference. It can take several days for NTP to fully synchronize the computer clock.

The DAG card clock is initialized from the computer's clock rather than from the NTP. Using NTP to synchronize the computer's clock ensures the DAG card clock remains accurate.

DAG cards can also be synchronized to external references such as GPS or to the computer clock directly. In both cases the computer clock time is loaded onto the DAG clock when the DAG card is started (`dagload, dagreset, dagrom -p`).

When clock synchronization is enabled, the DAG card time is compared to the computer time once per second, regardless of the synchronization source. If the times differ by more than 1 second, the DAG card clock is reloaded from the computer clock and synchronization is restarted. For this reason, the computer clock should be maintained with better than 1 second accuracy.

If the DAG card clock is synchronized to the computer clock, then small 'step' adjustments of the computer clock by the NTP daemon can cause the DAG driver to emit warning messages to the console and system log files if the adjustment exceeds the warning threshold. These messages are intended to allow the user to monitor the quality of the clock synchronization over time.

The best synchronization is achieved when the DAG card is synchronized to an external GPS reference clock, and the computer clock is synchronized to a local NTP server.

## Timestamps

ERF files contain a hardware generated timestamp of each packet's arrival.

The format of this timestamp is a single little-endian 64-bit fixed point number, representing the number of seconds since midnight on the 1st January 1970.

The high 32-bits contain the integer number of seconds, while the lower 32-bits contain the binary fraction of the second. This allows an ultimate resolution of $2^{-32}$ seconds, or approximately 233 picoseconds.

Different DAG cards have different actual resolutions. This is accommodated by the lower most bits that are not active being set to zero. In this way the interpretation of the timestamp does not need to change when higher resolution clock hardware is available. The DAG 3.7T implements the 27 most significant bits which provides a time resolution of 7.5 nanoseconds.

The ERF timestamp allows you to find the difference between two timestamps using a single 64-bit subtraction. You do not need to check for overflows between the two halves of the structure as you would need to do when comparing Unix time structures.

### Example

Below is example code showing how a 64-bit ERF timestamp (erfts) can be converted into a `struct timeval` representation (tv):

```
unsigned long long lts;
struct timeval tv;


  lts = erfts;
  tv.tv_sec = lts >> 32;
  lts = ((lts & 0xffffffffULL) * 1000 * 1000);
  lts += (lts & 0x80000000ULL) << 1;      /* rounding */
  tv.tv_usec = lts >> 32;
  if(tv.tv_usec >= 1000000) {
    tv.tv_usec -= 1000000;
    tv.tv_sec += 1;
      }
```

## Dagclock

The DUCK is very flexible and can be used with or without an external time reference. It can accept synchronization from one of several input sources and also be made to drive its synchronization output from one of several sources.

Synchronization settings are controlled by the `dagclock` utility.

**Note:** You should only run `dagclock` after you have loaded the appropriate FPGA images. If at any stage you reload the FPGA images you must rerun `dagclock` to restore the configuration.

**Note:** when you run `dagconfig -d0 default` the `dagclock` inputs and outputs are also reset to defaults.

A description of each argument is shown below:

| Option | Description |
|---|---|
| `-d DEVICE`<br>`--device DEVICE` | Use the DAG device referred to by DEVICE. Supported syntax includes 0, dag1, and /dev/dag3 to refer to DAG cards. |
| `-h, -?`<br>`--help, --usage` | Display the information on this page |
| `-k`<br>`--sync` | Wait for DUCK synchronization before exiting |
| `-K NUM` | Set the synchronization timeout in seconds (default is 60 seconds) |
| `-l NUM` | Set the Health threshold in nanoseconds. (default is 596ns) |
| `-v` | Increase output verbosity |
| `-V` | Display version information |
| `-x`<br>`--clearstats` | Clear clock statistics |

| Command | Description |
|---|---|
| `default` | Set the `dagclock` input and output to RS422 in and none out. |
| `none` | Clears the input and output settings. |
| `rs422in` | Sets the `dagclock` input to RS422. |
| `hostin` | Sets the `dagclock` input to Host (unused) |
| `overin` | Sets the `dagclock` input to Internal input |
| `auxin` | Sets the `dagclock` input to Auxiliary input (unused) |
| `rs422out` | Sets the `dagclock` output to repeat the RS422 input signal |
| `loop` | Output the selected input |
| `hostout` | Sets the `dagclock` output to host (unused) |
| `overout` | Internal output (master card) |
| `set` | Sets the DAG card's clock to computer clock time and clears clock statistics. The DAG card takes approximately 20 to 30 seconds to re-synchronize. |
| `reset` | Full clock reset. Load time from computer, set RS422in, none out. Clears clock statistics. The DAG card takes approximately 20 to 30 seconds to re-synchronize. |

**Note:** By default, all DAG cards listen for synchronization signals on their RS-422 port, and do not output any signal to that port.

## Dagclock Statistics reset

Statistics are reset to zero when the following occur:

- Loading a DAG driver
- Loading firmware
- `dagclock` with a `-x` option
- `dagclock` with a `set` or `reset` command.

### Example

To view the default `dagclock` configuration:

```
dagclock -d0
```

The following is the output from DAG card that has its clock reference connected. The clock statistics have been reset since the card was last synchronized. **Note:** Values will differ for each DAG card type.

```
muxin   rs422
muxout  none
status  Synchronised Threshold 596ns Failures 0 Resyncs 0
error   Freq -30ppb Phase -60ns Worst Freq 75ppb Worst Phase 104ns
crystal Actual 100000028Hz Synthesized 67108864Hz
input   Total 3765 Bad 0 Singles Missed 5 Longest Sequence Missed 1
start   Thu Apr 28 13:32:45 2007
host    Thu Apr 28 14:35:35 2007
dag     Thu Apr 28 14:35:35 2007
```

**Note:** For a description of the `dagclock` output see Dagclock output explained (page 70).

## Dagclock output explained

### Muxin

Lists the `dagclock` time input source for this DAG card. The options are `RS422in, Hostin, Overin or Auxin.`

Example
```
   muxin   rs422
```

### Muxout

Lists the `dagclock` time output source for this DAG card. The options are `RS422out, Hostout, Overout or Loop.`

Example
```
   muxout  none
```

### Status

This line reports on the status of the DAG card.

| Output | Description |
|---|---|
| **Synchronised / Not synchronised** | This indicates whether this DAG card is synchronized to the time source listed (`Muxin`) . The DAG card becomes **Not Synchronized** when the absolute *Phase error (page 70)* is above the *Threshold* value for 10 consecutive seconds. |
| **Threshold** | This is the value above which the DAG card port is considered **Not Synchronized**. The *Threshold* value changes depending on the type of input time synchronization. The defaults are: • 596 for RS422 synchronization • 12000 for host synchronization (Unix) • 50000 for host synchronization (Windows) This value can be adjusted using the `dagclock -l` option. |
| **Failures** | This is a count of the number of times the DAG card has become **Not Synchronized.** |
| **Resyncs** | This is a count of the number of times the DAG card Phase error has exceeded 1 second. See Error (Dagclock) (page 70). If the DAG card is **Not Synchronized** for more than 10 seconds the DAG card automatically runs the following command to update the time on the DAG card: `dagclock -d0 set` Where "0" is the device number. |

Example
```
   status  Synchronised Threshold 596ns Failures 0 Resyncs 0
```

### Error

This line reports on the synthesized frequency of the DAG card.

| Output | Description |
|---|---|
| Freq | An estimate of the synthesized frequency error over the last second in parts per billion. |
| Phase | The difference between the DAG card's clock and the reference clock at the last time pulse. |
| Worst Freq | Highest absolute value of the *Frequency error* since statistic collection began. Reset to zero when statistics are reset, see Dagclock Statistics reset (page 69). |
| Worst Phase | Highest absolute value of the *Phase error* since statistic collection began. Reset to zero when statistics are reset, see Dagclock Statistics reset. (page 69) |

Example
```
   error   Freq -30ppb Phase -60ns Worst Freq 75ppb Worst Phase 104ns
```

### Crystal

This line reports on the DAG card crystal oscillator.

| Output | Description |
|---|---|
| Actual | The DAG card's crystal frequency calculated based on the reference clock. |
| Synthesized | The target time stamping frequency. Different for each DAG card type. |

Example

```
crystal Actual 100000028Hz Synthesized 67108864Hz
```

### Input

This line reports on the time pulses received by the DAG card.

| Output | Description |
|---|---|
| Total | The total number of time pulses received. Reset to zero when statistics are reset, see Dagclock Statistics reset (page 69). |
| Bad | The number of time pulses that were rejected (considered Bad) by the DAG card. Reset to zero when statistics are reset, see Dagclock Statistics reset (page 69).<br>Time pulses are considered *Bad* if they were not received 1 second (approximately) after the last time pulse. These may be caused by noise. |
| Singles missed | The number of times a single time pulse failed to be received by the DAG card (i.e. a two second gap).<br>Reset to zero when statistics are reset, see Dagclock Statistics reset (page 69). |
| Longest Sequence Missed | This displays the longest time gap (in seconds) between a pair of time pulses. Reset to zero when statistics are reset, see Dagclock Statistics reset (page 69). |

Example

```
input   Total 3765 Bad 0 Singles Missed 5 Longest Sequence Missed 1
```

### Start / Host / DAG

| Output | Description |
|---|---|
| Start | This is the time statistics collection started. See Dagclock Statistics reset. (page 69) |
| Host | Current Host (computer) time. |
| DAG | The DAG card time at the last time pulse. If the DAG card has never been synchronized, the following displays:<br>`No active input - free running.` |

Example

```
start   Thu Apr 28 13:32:45 2007
host    Thu Apr 28 14:35:35 2007
dag     Thu Apr 28 14:35:35 2007
```

## Card with Reference

### Overview

To obtain the best timestamp accuracy you should connect the DAG card to an external clock reference, such as a GPS or CDMA time receiver.

To use an external clock reference source, the host computer's clock must be accurate to UTC to within one second. This is used to initialize the DUCK.

When the external time reference source is connected to the DAG card time synchronization connector, the DAG card automatically synchronizes to a valid signal.

### Pulse Signal from External Source

The DAG time synchronization connector supports an RS-422 (PPS) signal from an external source. This is derived directly from an external reference source or distributed through the Endace TDS 2 (Time Distribution Server) module which allows two DAG cards to use a single receiver.  It is also possible for more than two DAG cards to use a single receiver by "daisy-chaining" TDS-6 expansion modules to the TDS-2 module. Each TDS-6, module provides outputs for an additional 6 DAG cards.

Synchronize to an external source as follows:

```
dagclock -d0
```

Output:

```
muxin   rs422
muxout  none
status  Synchronised Threshold 596ns Failures 0 Resyncs 0
error   Freq 30ppb Phase -15ns Worst Freq 238ppb Worst Phase 326ns
crystal Actual 100000023Hz Synthesized 67108864Hz
input   Total 225 Bad 0 Singles Missed 1 Longest Sequence Missed 1
start   Thu Apr 28 14:55:20 2007
host    Thu Apr 28 14:59:06 2007
dag     Thu Apr 28 14:59:06 2007
```

### Connecting the Time Distribution Server

You can connect the TDS 2 module to the DAG card using DUCK crossover cable (page 76) (**Note:** A 4-pin to RJ45 adapter may be required). The TDS may be located up to 600m (2000ft) from the DAG card depending upon the quality of the cable used, possible interference sources and other environmental factors. Please refer to the *EDM05-01 Time Distribution Server User Guide* for more in formation.

**Caution:**

Never connect a DAG card and/or the TDS 2 module to active Ethernet equipment or telephone equipment.

### Testing the Signal

For Linux and FreeBSD, when a synchronization source is connected the driver outputs messages to the console log file `/var/log/messages`.

To test the signal is being received correctly and has the correct polarity use the `dagpps` tool as follows:

```
dagpps -d0
```

`dagpps` measures the input state many times over several seconds, displaying the polarity and length of input pulse.

## Single Card No Reference

When a single DAG card is used with no external reference, the DAG card can be synchronized to the host computer clock. Most computer clocks are not very accurate by themselves, but the DUCK drifts smoothly at the same rate as the computer clock.

The synchronization achieved with this method is not as accurate as using an external reference source such as GPS.

The DUCK clock is synchronized to a computer clock by setting input synchronization selector to overflow as follows:

```
dagclock –d0 none overin
```

Output

```
muxin   overin
muxout  none
status  Synchronised Threshold 11921ns Failures 0 Resyncs 0
error   Freq 1836ppb Phase 605ns Worst Freq 147ppb Worst Phase 324ns
crystal Actual 49999347Hz Synthesized 16777216Hz
input   Total 87039 Bad 0 Singles Missed 0 Longest Sequence Missed 0
start   Wed Apr 27 14:27:41 2007
host    Thu Apr 28 14:38:20 2007
dag     Thu Apr 28 14:38:20 2007
```

## Two Cards No Reference

### Overview

If you are using two DAG cards in a single host computer with no reference clock, you must synchronize the DAG cards using the same method if you wish to compare the timestamps between the two DAG cards. You may wish to do this for example if the two DAG cards monitor different directions of a single full-duplex link. You can synchronize the DAG cards in two ways:

- One DAG card can be a clock master for the second. This is useful if you want both DAG cards to be accurately synchronized with each other, but not so for absolute time of packet time-stamps, or
- One DAG card can synchronize to the host and also act as a master for the second DAG card.

### Synchronizing with Each Other

Although the master DAG card's clock drifts against UTC, the DAG cards will be locked together. This is achieved by connecting the time synchronization connectors of both DAG cards using a DUCK crossover cable (page 76) (**Note:** A 4-pin to RJ45 Adapter may be required).

Configure one of the DAG cards as the master so that the other defaults to being a slave as follows:

```
dagclock -d0 none overout
```

Output:

```
muxin   none
muxout  over
status  Not Synchronised Threshold 596ns Failures 0 Resyncs 0
error   Freq 0ppb Phase 0ns Worst Freq 213ppb Worst Phase 251ns
crystal Actual 100000000Hz Synthesized 67108864Hz
input   Total 0 Bad 0 Singles Missed 0 Longest Sequence Missed 0
start   Thu Apr 28 14:48:34 2007
host    Thu Apr 28 14:48:34 2007
dag     No active input - Free running
```

**Note:** The slave DAG card configuration is not shown as the default configuration will work.

## Synchronizing with Host

To prevent the DAG card clock time-stamps drifting against UTC, the master DAG card can be synchronized to the host computer's clock which in turn utilizes NTP. This provides a master signal to the slave DAG card.

Configure one DAG card to synchronize to the computer clock and output a RS-422 synchronization signal to the second DAG card as follows:

```
dagclock –d0 none overin overout
```

Output:

```
muxin   over
muxout  over
status  Synchronised Threshold 11921ns Failures 0 Resyncs 0
error   Freq -691ppb Phase -394ns Worst Freq 147ppb Worst Phase 424ns
crystal Actual 49999354Hz Synthesized 16777216Hz
input   Total 87464 Bad 0 Singles Missed 0 Longest Sequence Missed 0
start   Wed Apr 27 14:27:41 2007
host    Thu Apr 28 14:59:14 2007
dag     Thu Apr 28 14:59:14 2007
```

**Note:** The slave DAG card configuration is not shown, the default configuration is sufficient.

## Connector Pin-outs

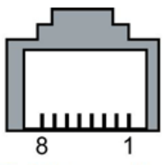
### Overview

DAG cards have an 8-pin RJ45 connector for time synchronization. The connector has two bi-directional RS422 differential circuits, A and B. The Pulse Per Second (PPS) signal is carried on circuit A, and the SERIAL packet is connected to the B circuit.

### Pin Assignments

The 8-pin RJ45 connector pin assignments and plugs and sockets are shown below:

| | |
|---|---|
| **1.** | Out PPS+ |
| **2.** | Out PPS- |
| **3.** | In PPS+ |
| **4.** | In SERIAL+ |
| **5.** | In SERIAL- |
| **6.** | In PPS- |
| **7.** | Out SERIAL+ |
| **8.** | Out SERIAL- |



RJ45 socket

Normally, you connect the GPS input to the PPS (A) channel input (pins 3 and 6).

The DAG card can also output a synchronization pulse for use when synchronizing two DAG cards (i.e. without a GPS input). The synchronization pulse is output on the Out PPS channel (pins 1 and 2).

To connect two DAG cards, use a DUCK crossover cable (page 76) to connect the two time synchronization sockets.

#### DUCK Crossover cable

To synchronize two DAG cards together use a cable with RJ45 plugs and the following wiring.

| | | | |
|---|---|---|---|
| TX PPS+ | **1** | **3** | RX PPS+ |
| TX PPS- | **2** | **6** | RX PPS- |
| RX PPS+ | **3** | **1** | TX PPS+ |
| RX SERIAL+ | **4** | **7** | TX SERIAL+ |
| RX SERIAL- | **5** | **8** | TX SERIAL- |
| RX PPS- | **6** | **2** | TX PPS- |
| TX SERIAL+ | **7** | **4** | RX SERIAL+ |
| TX SERIAL- | **8** | **5** | RX SERIAL- |

**Note:** This wiring is the same as an Ethernet crossover cable (Gigabit crossover, All four pairs crossed).

# Data Formats

## Overview

The DAG Card produces trace files in its own native format called ERF (Extensible Record Format). The ERF type depends upon the type of connection you are using to capture data.

The DAG 3.7T supports the following ERF Types:

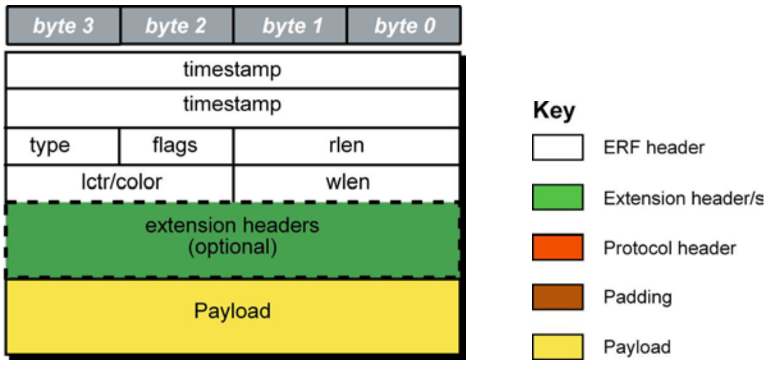
| ERF Type | Description |
|---|---|
| 5 | TYPE_MC_HDLC:<br>Multi-channel HDLC Frame Record |
| 6 | TYPE_MC_RAW<br>ERF Multi-Channel Raw Link Data Record. |
| 7 | TYPE_MC_ATM<br>Multi-Channel ATM Cell Record |
| 8 | TYPE_MC_RAW CHANNEL<br>Multi-Channel Raw Link Data Record |
| 9 | TYPE_MC_AAL5<br>Multi Channel AAL5 Frame Record |
| 12 | TYPE_MC_AAL2<br>Multi Channel AAL2 Frame Record |

The ERF file contains a series of ERF records with each record describing one packet. ERF files consists only of ERF records, there is no file header or trailer. This allows for simple concatenation and splitting of files to be performed on ERF record boundaries.

## Generic ERF Header

All ERF records share some common fields. Timestamps are in little-endian (Pentium® native) byte order. All other fields are in big-endian (network) byte order. All payload data is captured as a byte stream in network order, no byte or re-ordering is applied.

The generic ERF header is shown below:



The fields are described below:

| | | |
|---|---|---|
| timestamp | | The time of arrival of the cell, an ERF 64-bit timestamp. |
| type | Bit 7 | Extension header present. |
| | Bit 6:0 | Extension header type. See table below: |
| flags | | This byte is divided into several fields as follows: |
| | **Bits** | **Description** |
| | 1-0: | Binary enumeration of capture interface:<br>11 Interface 3 or D<br>10 Interface 2 or C<br>01 Interface 1 or B<br>00 Interface 0 or A<br>Cards with more than four interfaces typically use Multichannel ERF types (type 5 to 9, 12 and 17) which provide a separate larger interface field. |
| | 2: | Varying length record. When set, packets shorter than the snap length are not padded and rlen resembles wlen.<br>When clear, longer packets are snapped off at snap length and shorter packets are padded up to the snap length. rlen resembles snap length. Setting novarlen and slen greater than 256 bytes is wasteful of bandwidth |
| | 3: | Truncated record - insufficient buffer space.<br>• `wlen` is still correct for the packet on the wire.<br>• `rlen` is still correct for the resulting record. But, rlen is shorter than expected from snap length or wlen values.<br>**Note:** truncation is depreciated and this bit is unlikely to be set in an ERF record. |
| | 4: | RX error. An error in the received data. Present on the wire |
| | 5: | DS error. An internal error generated inside the card annotator. Not present on the wire. |
| | 6: | Reserved |
| | 7: | Reserved |
| rlen | | Record length in bytes. Total length of the record transferred over the PCI bus to storage.<br>The timestamp of the next ERF record starts exactly rlen bytes after the start of the timestamp of the current ERF record. |
| lctr | | Depending upon the ERF type this 16 bit field is either a loss counter of color field. The loss counter records the number of packets lost between the DAG card and the stream buffer due to overloading on the PCI bus. The loss is recorded between the current record and the previous record captured on the same stream/interface. The color field is explained under the appropriate type details. |

| | | |
|---|---|---|
| wlen | | Wire length. Packet length "on the wire" including some protocol overhead. The exact interpretation of this quantity depends on physical medium. This may contain padding. |
| extension headers | | Extension headers in an ERF record allow extra data relating to each packet to be transported to the host. Extension header/s are present if bit 7 of the type field is '1'. If bit 7 is '0', no extension headers are present (ensures backwards compatibility).<br>**Note:** There can be more than one Extension header attached to a ERF record. |
| Payload | | Payload is the actual data in the record. It can be calculated by either :<br>• Payload = rlen - ERF header - Extension headers (optional) - Protocol header - Padding |

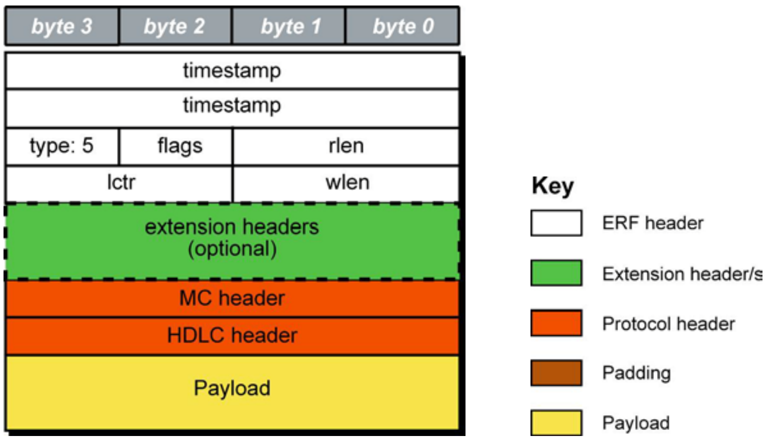
Extension header types

| Number | Type | Description |
|---|---|---|
| 0: | TYPE_LEGACY | Old style record |
| 1: | TYPE_HDLC_POS | Packet over SONET / SDH frames, using either PPP or CISCO HDLC framing. |
| 2: | TYPE_ETH | Ethernet |
| 3: | TYPE_ATM | ATM cell |
| 4: | TYPE_AAL5 | reassembled AAL5 frame |
| 5: | TYPE_MC_HDLC | Multi-channel HDLC frame |
| 6: | TYPE_MC_RAW | Multi-channel Raw time slot link data |
| 7: | TYPE_MC_ATM | Multi-channel ATM Cell |
| 8: | TYPE_MC_RAW_ CHANNEL | Multi-channel Raw link data |
| 9: | TYPE_MC_AAL5 | Multi-channel AAL5 frame |
| 10: | TYPE_COLOR_HDLC_ POS | HDLC format like TYPE_HDLC_POS, but with the LCNTR field reassigned as COLOR |
| 11: | TYPE_COLOR_ETH | Ethernet format like TYPE_ETH, but with the LCNTR field reassigned as COLOR |
| 12: | TYPE_MC_AAL2 | Multi-channel AAL2 frame |
| 13: | TYPE_IP_COUNTER | IP Counter ERF Record |
| 14: | TYPE_TCP_FLOW_ COUNTER | TCP Flow Counter ERF Record |
| 15: | TYPE_DSM_COLOR_ HDLC_POS | HDLC format like TYPE_HDLC_POS, but with the LCNTR field reassigned as DSM COLOR |
| 16: | TYPE_DSM_COLOR_ ETH | Ethernet format like TYPE_ETH, but with the LCNTR field reassigned as DSM COLOR |
| 17: | TYPE_COLOR_MC_ HDLC_POS | Multi-channel HDLC like TYPE_MC_HDLC, but with the LCNTR field reassigned as COLOUR |
| 18: | TYPE_AAL2 | Reassembled AAL2 Frame Record |
| 19: | TYPE_COLOR_HASH_ POS | Colored PoS HDLC record with Hash load balancing |
| 20: | TYPE_COLOR_HASH_ ETH | Colored Ethernet variable length record with Hash load balancing |
| 21: | TYPE_INFINIBAND | Infiniband Variable Length Record |
| 22: | TYPE_IPV4 | IPV4 Variable Length Record |
| 23: | TYPE_IPV6 | IPV6 Variable Length Record |
| 24 | TYPE_RAW_LINK | Raw link data, typically SONET or SDH Frame |
| 32-47: | - | Reserved for CoProcess Development Kit (CDK) Users and Internal use |
| 48: | TYPE_PAD | Pad Record type |

## ERF 5. TYPE_MC_HDLC

| | | |
|---|---|---|
| Type | Bit 7 | 1 = Extension header present. See Extension Headers (page 86). |
| | Bits 6:0 | Type 5 |
| Short description | TYPE_MC_HDLC | |
| Long description | Type 5 Multi-channel HDLC Frame Record | |
| Use | This record format is for channelized HDLC data links. For example E1, T1 and J1. | |

The *TYPE_MC_HDLC* record is shown below:



The following is a description of the *TYPE_MC_HDLC* record format:

| Field | Description |
|---|---|
| flags<br>(1 byte) | This field is the same as normal ERF types but capture interface is always zero.<br>• Fixed length mode not supported.<br>• RX Error is set if any MC Header Error bit is set. |
| MC header<br>(4 bytes) | Protocol Header. This field is divided into the following: (see table below) |
| HDLC header<br>(4 bytes) | Protocol header. Length may vary depending on protocol. |
| Payload<br>(bytes of packet) | Payload = rlen - ERF header (16 bytes) - Extension headers (optional) - Protocol header (8 bytes) |

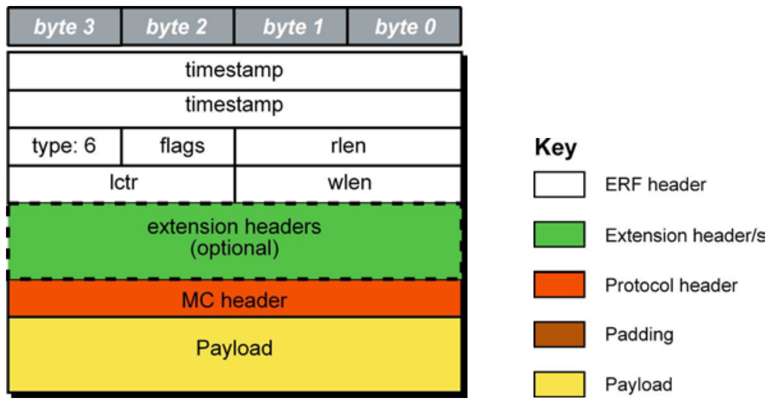
| Bits | Attribute |
|---|---|
| 0-9 | Connection Number [0-1023]. |
| 10-15 | Reserved. |
| 16-23 | Reserved. |
| 24 | FCS Error. |
| 25 | Short Record Error [<5 Bytes]. |
| 26 | Long Record Error [>2047 Bytes]. |
| 27 | Aborted Frame Error. |
| 28 | Octet Error. The closing flag was not octet aligned after bit stuffing. |
| 29 | Lost Byte Error. The internal data path had an unrecoverable error. |
| 30 | $1^{ST}$ Rec. This is the first record received since this connection was configured. |
| 31 | Reserved |

**Note**: When using this record type with the DAG 3.7T card the Interface number is 0, and the connection number is defined by the programmed context.
When using this record type with the DAG 7.1S card the interface number is used for the four ports, and the connection number is the VC identifier, as defined in the *EDM01-17 DAG 7.1S Card User Guide*.

## ERF 6. TYPE_MC_RAW

| | | |
|---|---|---|
| Type | Bit 7 | 1 = Extension header present. See Extension Headers (page 86). |
| | Bits 6:0 | Type 6 |
| Short description | TYPE_MC_RAW | |
| Long description | Type 6 Multi-Channel RAW Time Slot Link Data Record | |
| Use | This record format is for the RAW capture from data links. For example; E1, T1 and J1. | |

The *TYPE_MC_RAW* record is below:



The following is a description of the *TYPE_MC_RAW* record format:

| Field | Description |
|---|---|
| Flags (1 byte) | This field is the same as normal ERF types but capture interface is always zero. • Fixed length mode not supported. • RX Error is set if any MC Header Error bit is set. |
| MC header (4 bytes) | Protocol header. This field is divided into the following: (see table below) |
| Payload (bytes of raw link data) | Payload = rlen - ERF header (16 bytes) - Extension headers (optional) - Protocol header (4 bytes) This field is divided into the following: (see table below) |

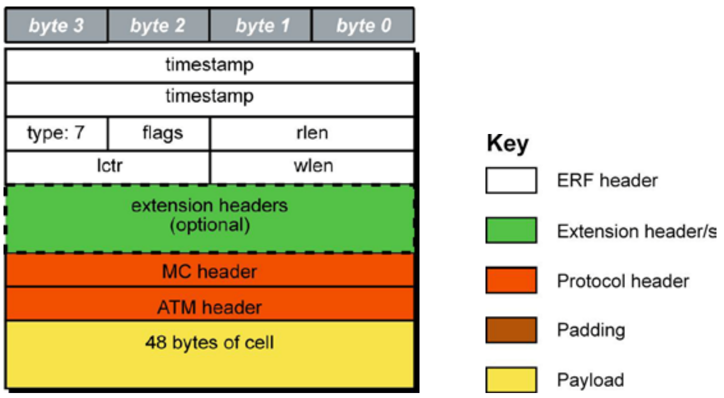
| Bits | Attribute |
|---|---|
| 0-3: | Physical Interface [0-15]. |
| 4-15: | Reserved. |
| 16-23: | Reserved. |
| 24: | Reserved. |
| 25: | Short Record [<6 Bytes]. |
| 26: | Long Record [>2047 Bytes] |
| 27: | Reserved. |
| 28: | Reserved. |
| 29: | Lost Byte. The internal datapath had an unrecoverable error. |
| 30: | 1st Rec. This is the first record received since this connection was configured. |
| 31: | Reserved. |

| Data type | Description |
|---|---|
| T1: | 24 bytes for 24 time slots. |
| E1: | 31 bytes for time slots 0-31. Slot 16 is signaling information. |
| Framed E1: | 30 bytes of data for time slots 1-31, slot 0 used for framing is not captured. Slot 16 is signaling information. |

## ERF 7. TYPE_MC_ATM

| | | |
|---|---|---|
| Type | Bit 7 | 1 = Extension header present. See Extension Headers (page 86). |
| | Bits 6:0 | Type 7 |
| Short description | TYPE_MC_ATM | |
| Long description | Type 7 Multi-channel ATM Cell Record | |
| Use | This record format is for ATM cells on channelized data links. For example; E1, T1 and J1. | |

The *TYPE_MC_ATM* record is shown below:



The following is a description of the *TYPE_MC_ATM* record format:

| Field | Description |
|---|---|
| flags (1 byte) | This field is the same as normal ERF types but capture interface is always zero. • Fixed length mode not supported. • RX Error is set if any MC Header Error bit is set. |
| MC header (4 bytes) | Protocol header. This field is divided into the following: |

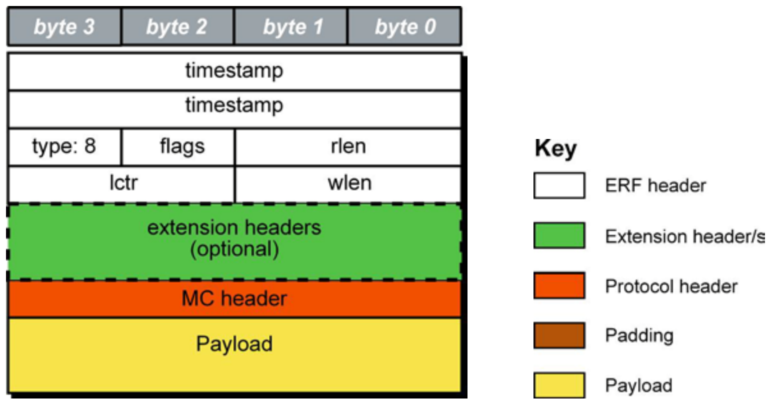
| Bit | Description |
|---|---|
| 0-9: | Connection number (0-1023). 512 connections are supported by DAG 3.7T card. For the DAG 7.1S card refer to *EDM01-17 DAG 7.1S Card User Guide* for details. Refer to the Channelized Configuration > Configuration File. |
| 10-14: | Reserved. |
| 15: | Multiplexed from IMA group into ATM stream. When bit 15 of the MC Header is set the bottom 9 bits (Connection Number/IMA ID) shall be treated as an IMA Group ID instead of a connection number. |
| 16-19: | Physical port [0-15] cell was captured on. Physical ID is interpreted from the firmware perspective. For example, if a cable is plugged into port 0, examining the ERF MC Header field will give a Physical ID of 11. This is a little counter-intuitive and reflects the internal processing required. From the software/user perspective, this could be interpreted as the Logical ID, and as such, we can convert from the Logical to Physical ID using the provided `dagutil` function, `dagutil_37t_line_get_logical` which will return the Software Physical ID/Firmware Logical ID. In this case, assuming data is coming in on a cable plugged into port 0, we will convert 11 back to 0. |
| 20-23: | Reserved. |
| 24: | Lost Byte. The internal datapath 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. |

| Field | Description |
|---|---|
| ATM header (4 bytes) | Protocol header. The ATM HEC channel is not captured. This record has a fixed length of 72 bytes. This does not include the 8-bit HEC. |
| Payload (bytes of cell) | Payload = 48 bytes of cell - HEC (1 byte) |

## ERF 8. TYPE_MC_RAW_CHANNEL

| | | |
|---|---|---|
| Type | Bit 7 | 1 = Extension header present. See Extension Headers (page 86). |
| | Bits 6:0 | Type 8 |
| Short description | TYPE_MC_RAW_CHANNEL | |
| Long description | Type 8 Multi-channel RAW Channel Multi-channel RAW Link Data Record | |
| Use | This record format captures complete RAW channelized data links. For example, E1, T1 and J1. | |

The *TYPE_MC_RAW_CHANNEL* record is shown below:



The following is a description of the *TYPE_MC_RAW_CHANNEL* record format:

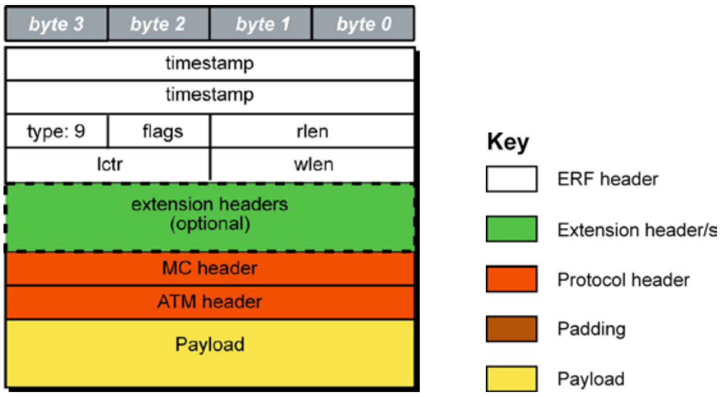
| Field | Description |
|---|---|
| flags<br>(1 byte) | This field is the same as normal ERF types but capture interface is always zero.<br>• Fixed length mode not supported.<br>• RX Error is set if any MC Header Error bit is set. |
| MC header<br>(4 bytes) | Protocol header. This field is divided into the following:<br>**Bits** / **Attributes**<br>0-9: Connection number (0-1023).<br>10-28: Reserved.<br>29: Lost Byte Error. The internal datapath had an unrecoverable error.<br>30: 1$^{st}$ Rec. This is the first record received since this connection was configured.<br>31: Reserved. |
| Payload<br>(bytes of data) | Payload = rlen - ERF header (16 bytes) - Extension headers (optional)<br>- Protocol header (4 bytes) |

**Note**: When using this record type with the DAG 3.7T card the Interface number is 0, and the connection number is defined by the programmed context.
When using this record type with the DAG 7.1S card the interface number is used for the four ports, and the connection number is the VC identifier, as defined in the DAG 7.1S Card User Guide.

## ERF 9. TYPE_MC_AAL5

| | | |
|---|---|---|
| Type | Bit 7 | 1 = Extension header present. See Extension Headers (page 86). |
| | Bits 6:0 | Type 9 |
| Short description | TYPE_MC_AAL5 | |
| Long description | Type 9 Multi-channel AAL5: Multi-channel AAL5 Frame Record | |
| Use | This record format for reassembled ATM AAL5 frames from channelized data links. For example; E1, T1, J1. | |

The *TYPE_MC_AAL5* record is shown below:



The following is a description of the *TYPE_MC_AAL5* record format:

| Field | Description |
|---|---|
| flags<br>(1 byte) | This field is the same as normal ERF types but capture interface is always zero.<br>• Fixed length mode not supported.<br>• RX Error is set if any MC. Header Error bit is set. |
| wlen<br>(2 bytes) | 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. |
| MC header<br>(4 bytes) | Protocol Header. This field is divided into the following: (see table below) |
| ATM header<br>(4 bytes) | Protocol Header. This does not include the 8-bit HEC. |
| Payload<br>(bytes of AAL5 frame) | Payload = rlen - ERF header (16 bytes) - Extension headers (optional) - Protocol header (8 bytes) |

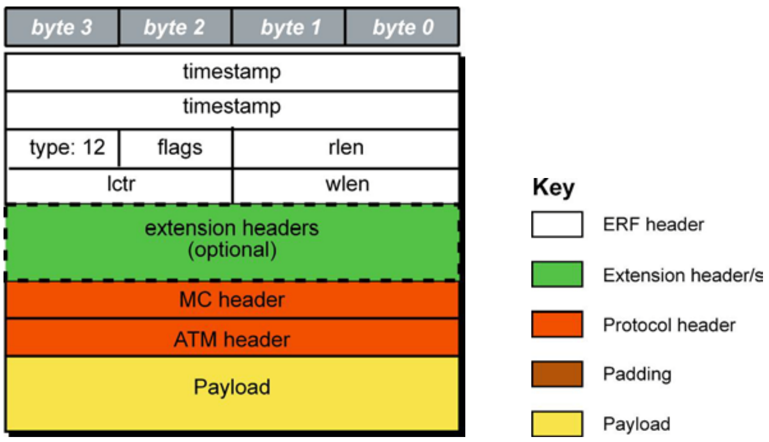
MC header bits:

| Bits | Attributes |
|---|---|
| 0-10: | Connection number (0-2047). 512 connections are supported by DAG 3.7T card. |
| 11-15: | Reserved. |
| 16-19: | Physical port (0-15) cell was captured on. Physical ID is interpreted from the firmware perspective. For example, if a cable is plugged into port 0, examining the ERF MC Header field will give a Physical ID of 11. This is a little counter-intuitive and reflects the internal processing required. From the software/user perspective, this could be interpreted as the Logical ID, and as such, we can convert from the Logical to Physical ID using the provided `dagutil` function, `dagutil_37t_line_get_logical` which will return the Software Physical ID/Firmware Logical ID. In this case, assuming data is coming in on a cable plugged into port 0, we will convert 11 back to 0. For the 7.1S this field is always 0. |
| 20: | CRC checked. |
| 21: | CRC error. |
| 22: | Length checked. |
| 23: | Length error. |
| 24-27: | Reserved. |
| 28: | 1st Cell. This is the first cell received since this connection was configured. |
| 29-31: | Reserved. |

## ERF 12. TYPE_MC_AAL2

| | | |
|---|---|---|
| Type | Bit 7 | 1 = Extension header present. See Extension Headers (page 86). |
| | Bits 6:0 | Type 12 |
| Short description | TYPE_MC_AAL2 | |
| Long description | Type 12 Multi-channel AAL25: Multi-channel AAL2 Frame Record | |
| Use | This record format is for channelized links is the same as the normal ERF Types but capture interface is always zero. | |

The *TYPE_MC_AAL2* record is shown below:



The following is a description of the *TYPE_MC_AAL2* record format:

| Field | Description |
|---|---|
| flags (1 byte1) | This field is the same as normal ERF types but capture interface is always zero.<br>• Fixed length mode not supported.<br>• RX Error is set if any MC Header Error bit is set. |
| MC header (4 bytes) | Protocol header. This field is divided into the following: (see table below) |
| ATM header (4 bytes) | Protocol header. This does not include the 8-bit HEC. |
| Payload (bytes of AAL5 frame) | Payload = rlen - ERF header (16 bytes) - Extension headers (optional) - Protocol header (8 bytes) |

| Bits | Attribute |
|---|---|
| 0-9 | Connection number (0-1023).<br>512 connections are supported by DAG 3.7T card. |
| 10-12 | Reserved for possible extra connection numbers |
| 13-15 | Reserved for indication of AAL2 type (a value of 0x0 indicates a SSSAR packet). |
| 16-19 | Physical port (0-15) cell was captured on.<br>Physical ID is interpreted from the firmware perspective. For example, if a cable is plugged into port 0, examining the ERF MC Header field will give a Physical ID of 11. This is a little counter-intuitive and reflects the internal processing required. From the software/user perspective, this could be interpreted as the Logical ID, and as such, we can convert from the Logical to Physical ID using the provided `dagutil` function, `dagutil_37t_line_get_logical` which will return the Software Physical ID/Firmware Logical ID. In this case, assuming data is coming in on a cable plugged into port 0, we will convert 11 back to 0. For the 7.1S this field is always 0. |
| 20 | Reserved |
| 21 | 1st Cell. This is the first cell received since this connection was configured. |
| 22 | MAAL Error (errnum as specified in ITU I.363.2 is copied to the data part of this record) |
| 23 | Length Error |
| 24-31 | Channel Identification Number (cid) |

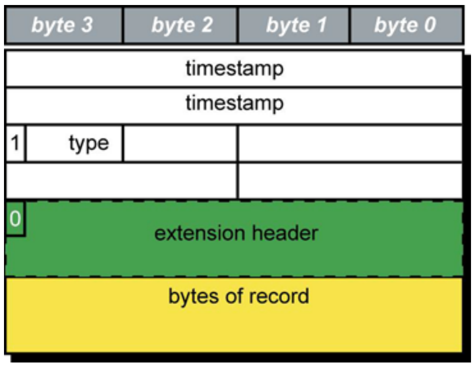
## Extension Headers (EH)

### Introduction

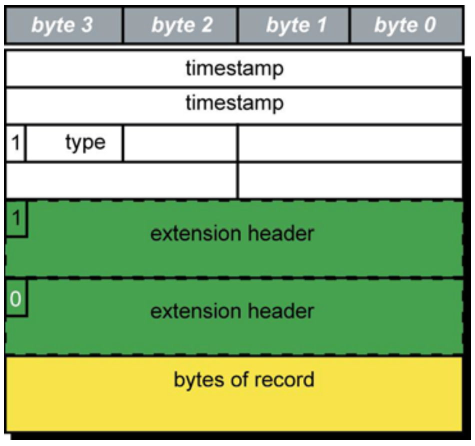
The addition of an Extension Header into the ERF record allows extra data relating to the packet to be transported to the host. The extension header allows certain features to be added independently of ERF types, for example, features shared by different ERF records do not have to be implemented separately. This results in automatic support across ERF types.

Bit 7 of the ERF type field is used to indicate that Extension Header's are present. If set to '1' Extension Headers are present. The Extension Header type field indicates the type and format of the Extension Header. It also indicates whether further Extension Headers are present. If bit 7 of the Extension Header is set to '1' further Extension Headers exist in the record. The Extension Headers are 8 bytes in length.

The following diagram shows presence of an Extension Header in addition to the ERF record.



The following diagram shows presence of two Extension Headers with Bit 7 of the first Extension Header set to '1'.



## Reporting Problems

If you have problems with a DAG 3.7T card or Endace supplied software which you are unable to resolve, please contact Endace Customer Support at support@endace.com.

Supplying as much information as possible enables Endace Customer Support to be more effective in their response to you. The exact information available to you for troubleshooting and analysis may be limited by nature of the problem.

The following items may assist in a quick resolution:

- DAG 3.7T card[s] model and serial number.
- Host computer type and configuration.
- Host computer operating system version.
- DAG software version package in use.
- Any compiler errors or warnings when building DAG driver or tools.
- For Linux and FreeBSD, messages generated when DAG device driver is loaded. These can be collected from command `dmesg`, or from log file `/var/log/syslog`.
- Output of `daginf`.
- Firmware versions from `dagrom -x`.
- Physical layer status reported by: `dagconfig`
- Link statistics reported by: `dagconfig -si`
- Statistics either (depending on the DAG card):
  - Extended statistics reported by: `dagconfig -ei`
  - Universal statistics reported by: `dagconfig -ui`
- Network link configuration from the router where available.
- Contents of any scripts in use.
- Complete output of session where error occurred including any error messages from DAG tools. The `typescript` Unix utility may be useful for recording this information.
- A small section of captured packet trace illustrating the problem.
- If you have just rebooted and the system can not see any DAG cards, you need to load the DAG drivers. Run `dagload`.

# Version History

| Version | Date | Reason |
|---|---|---|
| 1-9 | Pre 2006 | Old Versions |
| 10 | May 2006 | First version new format, major review of existing document |
| 11 | August 2006 | Added version history<br>Extended functions new mixed image file system information.<br>Added generic DAG information to: Inserting the DAG Card and Transmitting.<br>Added new section Transmitting Packets to HDLC and ATM configuration<br>Alternate highlighting changes and proofing |
| 12 | December 2006 | General editing and formatting changes.<br>Replacement of LED diagrams with photo<br>Addition of IMA transmit information |
| 13 | April 2007 | Addition of Pod Chassis mounting information. |
| 14 | November 2007 | Merged into AuthorIT.  Added additional information. |
| 15 | January 2008 | Updates and changes to configuring section, general revision. |
| 16 | June 2008 | Updated to dagconfig.  Updated tokens list.  Added card features and firmware image list.  Updated statistics.  Updated Overview section.  Depreciated dagthree.  Added ERF Extension header information. |
| 17 | August 2008 | Released.  Added loss values to the mode information.  Added caution about high gain modes affect on configured but unconnected interfaces. |
| 18 | November 2008 | Updated Buffer_size and mem dagconfig tokens and associated cross references.  Updated front matter. Update dagconfig options table.  Added new dagrom options.  Supported OS information now in release notes.  Added card description to the overview.  Updated Pod Racket mount information. |

| Status | Description |
|---|---|
| Preliminary | The products described in this technical document are in development and have yet to complete final production quality assurance. |
| Released | The products described in this technical document have completed development and final production quality assurance. |