



Addendum

Camera Configuration Guide for Camera Link cameras



**(when used in conjunction with
EDT digital video products)**

**Doc. 008-01993-04
Rev. 2011 August 23**

Engineering Design Team (EDT), Inc.

1400 NW Compton Drive, Suite 315

Beaverton, OR 97006

p 503-690-1234 / 800-435-4320

f 503-690-1243

www.edt.com

EDT™ and Engineering Design Team™ are trademarks of Engineering Design Team, Inc. All other trademarks, service marks, and copyrights are the property of their respective owners†.

© 1997-2011 Engineering Design Team, Inc. All rights reserved.

Terms of Use Agreement

Definitions. This agreement, between Engineering Design Team, Inc. (“Seller”) and the user or distributor (“Buyer”), covers the use and distribution of the following items provided by Seller: a) the binary and all provided source code for any and all device drivers, software libraries, utilities, and example applications (collectively, “Software”); b) the binary and all provided source code for any and all configurable or programmable devices (collectively, “Firmware”); and c) the computer boards and all other physical components (collectively, “Hardware”). Software, Firmware, and Hardware are collectively referred to as “Products.” This agreement also covers Seller’s published Limited Warranty (“Warranty”) and all other published manuals and product information in physical, electronic, or any other form (“Documentation”).

License. Seller grants Buyer the right to use or distribute Seller’s Software and Firmware Products solely to enable Seller’s Hardware Products. Seller’s Software and Firmware must be used on the same computer as Seller’s Hardware. Seller’s Products and Documentation are furnished under, and may be used only in accordance with, the terms of this agreement. By using or distributing Seller’s Products and Documentation, Buyer agrees to the terms of this agreement, as well as any additional agreements (such as a nondisclosure agreement) between Buyer and Seller.

Export Restrictions. Buyer will not permit Seller’s Software, Firmware, or Hardware to be sent to, or used in, any other country except in compliance with applicable U.S. laws and regulations. For clarification or advice on such laws and regulations, Buyer should contact: U.S. Department of Commerce, Export Division, Washington, D.C., 20230, U.S.A.

Limitation of Rights. Seller grants Buyer a royalty-free right to modify, reproduce, and distribute executable files using the Seller’s Software and Firmware, provided that: a) the source code and executable files will be used only with Seller’s Hardware; b) Buyer agrees to indemnify, hold harmless, and defend Seller from and against any claims or lawsuits, including attorneys’ fees, that arise or result from the use or distribution of Buyer’s products containing Seller’s Products. Seller’s Hardware may not be copied or recreated in any form or by any means without Seller’s express written consent.

No Liability for Consequential Damages. In no event will Seller, its directors, officers, employees, or agents be liable to Buyer for any consequential, incidental, or indirect damages (including damages for business interruptions, loss of business profits or information, and the like) arising out of the use or inability to use the Products, even if Seller has been advised of the possibility of such damages. Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to Buyer. Seller’s liability to Buyer for actual damages for any cause whatsoever, and regardless of the form of the action (whether in contract, product liability, tort including negligence, or otherwise) will be limited to fifty U.S. dollars (\$50.00).

Limited Hardware Warranty. Seller warrants that the Hardware it manufactures and sells shall be free of defects in materials and workmanship for a period of 12 months from date of shipment to initial Buyer. This warranty does not apply to any product that is misused, abused, repaired, or otherwise modified by Buyer or others. Seller’s sole obligation for breach of this warranty shall be to repair or replace (F.O.B. Seller’s plant, Beaverton, Oregon, USA) any goods that are found to be non-conforming or defective as specified by Buyer within 30 days of discovery of any defect. Buyer shall bear all installation and transportation expenses, and all other incidental expenses and damages.

Limitation of Liability. *In no event shall Seller be liable for any type of special consequential, incidental, or penal damages, whether such damages arise from, or are a result of, breach of contract, warranty, tort (including negligence), strict liability, or otherwise.* All references to damages herein shall include, but not be limited to: loss of profit or revenue; loss of use of the goods or associated equipment; costs of substitute goods, equipment, or facilities; downtime costs; or claims for damages. Seller shall not be liable for any loss, claim, expense, or damage caused by, contributed to, or arising out of the acts or omissions of Buyer, whether negligent or otherwise.

No Other Warranties. Seller makes no other warranties, express or implied, including without limitation the implied warranties of merchantability and fitness for a particular purpose, regarding Seller’s Products or Documentation. Seller does not warrant, guarantee, or make any representations regarding the use or the results of the use of the Products or Documentation or their correctness, accuracy, reliability, currentness, or otherwise. All risk related to the results and performance of the Products and Documentation is assumed by Buyer. The exclusion of implied warranties is not permitted by some jurisdictions. The above exclusion may not apply to Buyer.

Disclaimer. Seller’s Products and Documentation, including this document, are subject to change without notice. Documentation does not represent a commitment from Seller.

Contents

Camera Configuration Guide	
Overview	1
Camera Configuration Files	1
Initialization Tools.....	1
Related Resources.....	2
Creating a Camera Configuration File	3
Required Directives.....	4
camera_class, camera_model, camera_info	4
extdepth, depth, width, height	4
Clipping and Padding.....	5
Data Format	5
Deinterleaving and Other Data Reordering Schemes.....	6
Mode Control (CC) Lines	6
Decoding Bayer-filtered Images.....	6
Headers and Footers	7
Utilities	7
Using initcam.....	8
Using countbits.....	8
Using checkcam.....	10
Directives	12
Revision Log.....	38

Camera Configuration Guide

Overview

Before your EDT digital video board can recognize and access a camera, registers on the board and variables in the driver must be initialized with the proper settings for your camera model and operating mode. This initialization can be achieved via the camera configuration files and initialization tools below, which are all provided in your EDT installation package. In that package, the subdirectory `camera_config` contains camera configuration (`.cfg`) files, in editable ASCII text, for all cameras tested by EDT.

This guide is intended to help people running setups for which there is no EDT-provided camera configuration file. If your setup is one of those, this guide will help you create the file you need.

Camera Configuration Files

In the `camera_config` subdirectory, each file enables a particular camera model and operating mode, so each supported camera may have several files depending on its EDT-supported modes. For example, one file may initialize your camera model for full-frame mode, while another may do so for 2x2 binned mode.

In these files, each line is either a directive (a `directive: value` pair) or a comment (in which case it begins with `#`). Each camera typically requires only a few of the defined directives for proper operation.

To find the correct file for your camera and operating mode, you can search the file names using a Unix-style (or similar) `grep` function. If there is no file for your setup, simply create your own configuration file, as described in [Creating a Camera Configuration File on page 3](#).

Initialization Tools

Once you have found or created an appropriate camera configuration file, you can initialize your EDT board and driver by invoking the utility `initcam`, with an argument specifying the appropriate configuration file. `initcam` reads the file using the library subroutine `pdv_readcfg` in the EDT application programming interface (API), and then calls `pdv_initcam` to set the registers and variables based on the settings in the file. In addition, `pdv_initcam` can send commands to the camera to put it into a known state.

Alternatively, you can perform initialization using tools other than `initcam`. For example:

- `PdvShow` is an EDT capture and display application with a graphical user interface (GUI). When you first run `PdvShow`, a Camera Setup dialog first asks you to select your camera configuration; it then creates a script that runs `initcam`, invoking the appropriate configuration file.
- Optionally, initialize using library calls to subroutines `pdv_readcfg` and `pdv_initcam`. For an example of how to incorporate initialization codes into your application, see the `initcam.c` source code.

Related Resources

The resources below may be helpful or necessary for your applications.

- To find complete details on any EDT product, go to www.edt.com and find the appropriate product page. That page will provide links to the product's datasheet specifications and user's guide.
- To find EDT information that is not related to a specific EDT product (such as installation packages, or cable pinouts that apply to multiple products), go to www.edt.com and look in Product Documentation.

EDT Resources

	Detail	Web link
• Application programming interface (API)	HTML and PDF versions	www.edt.com/manuals.html
• Installation packages (Windows, Linux, etc.)	Software / firmware	www.edt.com/software.html
• Documentation for individual EDT products	Datasheets / user's guides	www.edt.com (find product page)
• IRIG details	Time Distribution user's guide	www.edt.com (find product page)
• Addendum – Camera Link firmware	Camera Link firmware setup	www.edt.com/manuals/PDV/dv-clink.pdf
• Addendum – non-Camera Link firmware	AIA firmware setup	www.edt.com/manuals/PDV/aiaag_add_dv2.pdf
• Pinouts – Camera Link	All modes (base, medium, full)	www.edt.com/manuals/PDV/cameraLinkPinout.pdf
• App note – Redlake/DuncanTech	For Redlake/Duncan cameras	www.edt.com/manuals/Cameras/duncan.txt

Standard / Specification For

		From	Web link
• PCI / PCIe	PCI / PCIe bus	PCI Special Interest Group (PCI SIG)	www.pcisig.com
• Camera Link	Camera Link	Machine Vision Online (MVO)	www.machinevisiononline.org
• IRIG-B	IRIG-B timestamping	Inter Range Instrumentation Group mod B	irigb.com

Creating a Camera Configuration File

If there is no file for your setup, you can copy an existing EDT file, edit it for your needs, rename and resave it, and use it as you would the original EDT file.

To create a camera configuration file:

1. Find a file for a setup that is similar to yours, or use one of these generic files (with the numbers 8, 10, 12, and 24 representing the number of bits):
 - Camera Link (monochrome): `generic8cl.cfg`, `generic10cl.cfg`, `generic12cl.cfg`, `generic14cl.cfg`, `generic16cl.cfg`
 - Camera Link (RGB color): `generic24cl.cfg`
 - Camera Link (Bayer color): `generic24bcl.cfg`
 - Non-Camera Link: `generic8.cfg`, `generic10.cfg`, `generic12.cfg`.
2. Make a copy of an appropriate `.cfg` file, or create a new file with a `.cfg` extension; then open your new file up in a text editor such as WordPad, eMacs, or VI.

NOTE Notepad has problems displaying Unix-style text files that do not have CR/LF line terminators, so if you have trouble viewing or editing the provided files in Notepad, try another editor such as WordPad.

3. Add or modify the camera ID string directives `camera_class`, `camera_model`, and `camera_info` to create a unique identifier for your new file.
4. Add or modify the image size and depth directives to match the camera's output in the areas of exact image size, lines per frame, and bits per pixel.
5. Add or modify the timing and data order directives to match the camera's output format.
 - For Camera Link, these directives are the following: `CL_DATA_PATH_NORM` and `CL_CFG_NORM`.
 - For non-Camera Link, these directives are some or all of the following: `shift`, `mask`, `byteswap`, `shortswap`, `DUAL_CHANNEL`, and `DOUBLE_RATE`.
6. If necessary, add the data reordering directive `method_interlace`.
7. If necessary, add region-of-interest or hardware padding directives to clip or pad the pixels per line to a four-byte boundary, and optionally clip off invalid data on the borders.
8. If you need board-controlled timing for triggering or exposure, set the directives `MODE_CNTL_NORM` and `method_camera_shutter_timing` to put the board into the desired mode.
9. Optionally, add a serial initialization directive (`serial_init` or a related directive) to put the camera into the desired state or mode on startup.
10. Make other edits as needed (see [Directives on page 12](#)).

Required Directives

The directives required for your setup will depend on which camera and EDT framegrabber you use.

These directives are required for all setups:

```
camera_class
camera_model
camera_info
width
height
depth
extdepth
```

These additional directives are required for all Camera Link setups:

```
CL_CFG_NORM
CL_DATA_PATH_NORM
```

These additional directives are required for setups with EDT's PCI DV FOX, PCI DVa, PCI DV, or PCI DVK:

```
rbtfile
```

For details, and for other (optional) directives, see [Directives on page 12](#).

camera_class, camera_model, camera_info

Typically, the directives `camera_class`, `camera_model`, and `camera_info` are used as shown below.

```
camera_class: (normally the camera manufacturer – for example, "Redlake")
camera_model: (normally the camera model – for example, "1310C")
camera_info: (normally a specific operating mode – for example, "10-bit monochrome")
```

extdepth, depth, width, height

Initially, set `extdepth` and `depth` to the number of bits per pixel in the camera output. However:

- If you have a 10- to 16-bit monochrome camera from which you wish to pass only 8 bits, set `depth` to 8 and `extdepth` to the number of bits per pixel from the camera.
- If you have a color RGB or Bayer-filtered camera, set `depth` to 24 and `extdepth` to the number of bits per red, green, or blue element (usually 8, 10 or 12).

For `width` and `height`, you may find that your camera outputs a slightly different number of lines per image, or pixels per line, than the documentation states (because the active pixel area of the CCD is sometimes greater than specified in the camera documentation). If this is true for your camera, you may need to use the trial-and-error method to find the correct values for `width` and `height`.

- Diagonal skew usually indicates a problem with `width`.
- Timeouts or overruns (reported in `PdvShow` at the bottom of the pane, or in the `take` application as a `timeouts:` or `overruns:` line) may indicate a problem with `height` or possibly `width`.

To find out what a free-running camera is outputting to PCI DV and PCI DVK boards, you can use `checkcam` (see [Using checkcam on page 10](#)).

Clipping and Padding

If desired, you can use region-of-interest directives to clip off unwanted borders in the incoming images. One reason you might do so is that many display programs (including `PdvShow`) use the Windows MFC display library, which is optimized for images aligned to a four-byte width. To avoid display problems with these programs, the number of pixels per line must be a multiple of four (for 8-bit cameras) or two (for 10- through 16-bit cameras).

If you cannot get the data to line up vertically, regardless of the `width` value you use, you may need to clip or pad the data to align on four-byte boundaries. To do so, you can use the hardware region-of-interest directives `hskip` and `hactv` to clip the data to a four-byte aligned size, or to clip the black inactive borders that many cameras send, or to do both. The `hskip`, `hactv`, `vskip`, and `vactv` directives activate PCI DV hardware-level clipping, eliminating the performance issues caused by software manipulations.

For example, suppose a camera with 1024 x 1024 active pixels also sends inactive pixel data before and after each line or frame, yielding a 1038 x 1038 image. Assuming equal borders of invalid data all around, you can clip off invalid data and get a four-byte aligned image (1024 x 1024 pixels) by using these directives:

```
hskip: 7
hactv: 1024
vskip: 7
vactv: 1024
```

NOTE This method will not work with some data reordering methods, as reordering assumes sequential data.

For details on the above directives, see [Directives on page 12](#).

Data Format

For Camera Link devices: The directive `CL_DATA_PATH_NORM` controls the expected number of bits per pixel and Camera Link taps. The argument is a hexadecimal byte where the left nibble is the number of taps minus 1, and the right nibble is the number of bits per pixel minus one.

For example:

```
CL_DATA_PATH_NORM: 07 # [1 tap, 8 bits]
CL_DATA_PATH_NORM: 1b # [2 taps, 12 bits]
```

The directive `CL_CFG_NORM` is used to set various other Camera Link-specific attributes, such as linescan, external triggering, and DVAL usage. The most typical setting is 02.

For example:

```
CL_CFG_NORM: 02
```

For other possible settings, see [Directives on page 12](#).

If your data is noisy, grainy, or banded, the configuration file may need the `shift` or `mask` directive, or both; or the argument in `CL_DATA_PATH_NORM` may be wrong. In such cases, you may wish to count the number of bit toggles for each bit in a raw image and output the result (see [Using countbits on page 8](#)).

For non-Camera Link devices: The directives `shift`, `mask`, `byteswap`, `shortswap`, `DOUBLE_RATE`, and `DUAL_CHANNEL` control how the data is manipulated before being sent to system memory using DMA. If these directives are not present in the configuration file, `initcam` defaults to values that match a typical AIA-swapped camera such as the Redlake MEGAPLUS series. If `byteswap` is not set, `initcam` sets it to 0 on little-endian systems (such as Intel), or 1 on big-endian systems (such as Sun).

EDT recommends omitting the directives `shift`, `mask`, `byteswap`, `shortswap`, `DOUBLE_RATE`, and `DUAL_CHANNEL`. However, if your data is grainy, noisy, or banded, the configuration file may need the `shift` or `mask` directive, or both. In such cases, you may wish to count the number of bit toggles for each bit in a raw image and output the result (see [Using countbits on page 8](#)).

Deinterleaving and Other Data Reordering Schemes

With some cameras, data must be reordered before it can be displayed. Several reordering algorithms are included in the EDT API (see [Related Resources on page 2](#)). The `method_interlace` directive causes the data acquisition subroutines (for example, `pdv_wait_image`) to call the specified algorithm before passing the image to a display routine. For example, a Redlake MEGAPLUS 8-bit camera running in dual-channel mode sends the data in pixel pairs, with one pixel from an odd line and one from an even line.

For this process, the deinterleave method is `BYTE_INTLV`, and the directive to enable reordering is:

```
method_interlace: BYTE_INTLV
```

See [method_interlace](#) for a description of the available reordering methods.

Mode Control (CC) Lines

Most cameras power up in free-run mode, so the board gets the next image on an acquire request. If, however, you need board-controlled triggering or exposure timing, use the mode control (CC) lines.

A few cameras also use mode control lines to control such things as gain, black level and binning, employing a variety of schemes. If you have a triggered camera and a cable that is wired properly, setting the `MODE_CNTL_NORM` directive to 10 is often all that is needed to enable triggering from the board. For other schemes, see [MODE_CNTL_NORM](#) and consult your camera documentation.

Decoding Bayer-filtered Images

To enable Bayer decoding in the library so that you can produce RGB data from Bayer-filtered data:

1. Copy a working monochrome camera configuration file and save it under a new name.
2. Change the `camera_info:` directive to include information specifying that this configuration file decodes a Bayer-filtered image.
3. Change `depth` to 24 (but do not change `extdepth`).
4. Add the following directives:

```
method_interlace: BGGR #for 8-bit data
method_interlace: BGGR_WORD #for 10- to 16-bit data
kbs_red_row_first: 0
kbs_green_pixel_first: 0
```

5. Try configuring the board with the new configuration file and acquiring an image with the camera.

If you did not set the values for `kbs_red_row_first` and `kbs_green_pixel_first` to match the sensors' filter layout, the colors will look wrong. To correct them, try different combinations of values for those directives until the colors look nearly correct, indicating that the camera configuration matches the camera sensor. However, the colors will not look precisely correct until you set the white balance.

6. To set the white balance, invoke `pdvshow`.

- a. On the toolbar, click the color wheel icon.
- b. Place something white, such as a white piece of paper, in front of the lens.
- c. Click **Compute white balance**.

The image now should appear correct.

For information on enabling and controlling Bayer decoding and white balance from your application, review `pdv_set_full_bayer_parameters` in the EDT API (see [Related Resources on page 2](#)).

Headers and Footers

Rarely, a camera may output an extra field of data in the header (before the image data) or in the footer (after the image data), or both. If the data is a multiple of the line width, you can increase the height to include the extra data, and then clip it off if you wish.

In unusual cases, the extra data may not be a multiple of the line width, so the count (from `checkcam`) will not be a multiple of $(\text{lines per frame}) * (\text{pixels per line})$.

Extra data at the end of the frame is not likely to cause a problem. However, extra data at the beginning of the frame may appear as an image that is uniformly shifted to the right. In this case, you should add header directives so that the board will read the extra data and store it in a header before going on to the frame data itself. To do so, follow the steps below.

1. Determine how much extra data must be stored as header, by computing the remainder of...

```
count / bytes_per_frame
```

...where `count` is the output from `checkcam`, and `bytes_per_frame` is defined as...

$$\frac{\text{lines}}{\text{frame}} \times \left(\frac{\text{pixels}}{\text{line}} \times \frac{\text{bytes}}{\text{pixel}} \right)$$

2. Use the result as the value of the `header_size` argument. For example:

```
method_header_position: HEADER_WITHIN
header_size: 398
header_dma: 1
```

For details on header and footer directives, see [Directives on page 12](#).

Utilities

Your EDT installation package contains utilities to initialize the EDT device for your camera, and to help you verify that your equipment is configured and operating correctly. They are:

- `initcam` – to initialize the device.
- `countbits` – to determine whether the image data is properly aligned.
- `checkcam` – to determine the number of pixels per line and lines per frame output from a free-running camera.

These utilities are described below.

Using `initcam`

In order for a newly created camera configuration file to take effect, you must run `initcam` with the new file (since `PdvShow` does not run `initcam` every time it is started). Then you can use a GUI capture and display application such as `PdvShow`, or an application of your own, to check your results.

For example:

```
initcam -f camera_config/file.cfg
```

When you select a camera configuration via `pdvshow`, the driver creates the file `camsetup0_0.bat` (on Windows) or `pdvload` (on UNIX-based systems). This file is a script that runs `initcam` with your selected configuration file as an argument.

Using `countbits`

The `countbits` utility is designed to help you determine whether image data is properly aligned. It takes a raw image file and counts the number of times each bit in a pixel changes value, compared to the value of the corresponding bit in the previous pixel. It then outputs the results of this count.

A properly aligned image ordinarily exhibits a descending number of changes in value from the least to the most significant bits of adjacent pixels. That is, the least significant bits of pixels are ordinarily the most volatile, changing values quite often to reflect small fluctuations of luminance, or color, as compared with adjacent pixels. These small fluctuations are often the result of noise in the sensor or tiny changes in the way light falls on areas of the subject. However, changes to more significant bits generally indicate larger changes in luminance or color — not a result of random noise in the sensor or the lighting, but rather changes in actual luminance or color value of the image subject. Such changes will therefore tend to occur less often. An image with properly aligned pixel data therefore shows a great many value changes in less significant bits and relatively fewer changes in more significant bits, forming a gradient of value changes, from many changes in the least significant bit to few changes in the most.

Results other than this descending gradient generally indicate a problem: the cable might be wired incorrectly or not wired for straight-through data, or the camera configuration might be in error in some way — an incorrect `CL_CFG_NORM` setting (for Camera Link cameras) or `shift`, `mask`, `byteswap`, or `shortswap` settings (for non-Camera Link cameras). In this case, viewing the image in an application such as `PdvShow` displays an image that appears snowy or obviously wrong.

To use `countbits`:

1. Run `take` to acquire and save a raw image

```
take -f file.raw
```

2. To run `countbits` on a monochrome 8-bit image:

```
countbits file.raw
```

For 10- to 16-bit monochrome images, use the `-w` option:

```
countbits -w file.raw
```

For RGB images with eight bits each of red, green, and blue information per pixel, use the `-c` option:

```
countbits -c file.raw
```

For monochrome 8-bit images, a properly aligned image yields results similar to these:

```
bit 00: 1061090
bit 01: 569471
bit 02: 247656
bit 03: 156286
bit 04: 71844
bit 05: 44572
bit 06: 22244
bit 07: 15588
```

For 10- to 16-bit monochrome images, a properly aligned image yields results similar to these:

```
bit 00: 248681
bit 01: 248573
bit 02: 248453
bit 03: 244508
bit 04: 230606
bit 05: 183557
bit 06: 121598
bit 07: 71613
bit 08: 40028
bit 09: 17169
bit 10: 0
bit 11: 0
bit 12: 0
bit 13: 0
bit 14: 0
bit 15: 0
```

For 24-bit RGB images, a properly aligned image yields results similar to these, showing three different gradients (one each for red, green, and blue):

Blue:

```
bit 00: 68375
bit 01: 39681
bit 02: 23249
bit 03: 14904
bit 04: 4868
bit 05: 2299
bit 06: 1212
bit 07: 270
```

Green:

```
bit 00: 104357
bit 01: 104938
bit 02: 62374
bit 03: 37419
bit 04: 15010
bit 05: 5062
bit 06: 2481
bit 07: 1148
```

```

Red:
bit 00: 73512
bit 01: 44331
bit 02: 26619
bit 03: 14443
bit 04: 4966
bit 05: 2877
bit 06: 1478
bit 07: 340

```

If your results are notably different from those above, analyze the data for clues as to what the problem is.

Using checkcam

Use the `checkcam` utility to determine the number of pixels per line and lines per frame output from a free-running camera.

NOTE `checkcam` runs only on PCI DV and PCI DVK boards, not on PCI DVa or any Camera Link board. For Camera Link, instead run `setdebug -d 0` and look at the hexadecimal `LINESPERFRAME` and `PIXELSPERLINE` counters. For details, see `setdebug` in the user's guide for EDT framegrabbers (see [Related Resources on page 2](#)).

To use `checkcam`:

1. Put your camera into continuous capture mode. If your camera runs in continuous capture mode by default, skip to [step 2](#). If your camera uses a serial protocol to switch between capture modes, use the `serial_cmd` utility to put the camera into continuous mode. Before doing so, follow these steps:
 - a. run a camera configuration file (existing or created) to initialize the board and driver.
 - b. Use `serial_cmd` to send the command. For example, for a Redlake MASD camera, use:

```
serial_cmd "MDE CS"
```

For special commands or mode lines for your camera, consult your camera documentation.

2. Enter the following at the command line:

```
initcam -f camera_config/cameratest.cfg
checkcam
```

3. When you see repeating lines like these, press your operating system interrupt to break out of the cycle:

```

waiting for frame valid
waiting for not frame valid
frame 1 count 614400 bad 0 lines/frame 480 pixels/line 640 dtime 0.084
frame 2 count 614400 bad 0 lines/frame 480 pixels/line 640 dtime 0.084
frame 3 count 614400 bad 0 lines/frame 480 pixels/line 640 dtime 0.084
. . . . .
. . . . .
. . . . .

```

NOTE If you do not get results similar to those shown above, then the camera is not in continuous output mode or there is a signal problem. Ensure that the camera is outputting good Frame Valid, Line Valid and Pixel clock signals, and the cable pinout is correct. Refer to the AIA specification, and check the signal levels.

4. In your camera configuration file, replace the `width` and `height` values with the respective values for number of pixels per line and number of lines per frame.

If the camera cannot be put into continuous mode (or cannot be triggered while running the `checkcam` utility), use the trial-and-error method to find the problem:

1. Substitute the `width` and `height` arguments in your test camera configuration file with the published pixels per line and lines per frame for the camera.
2. Run an application program such as `PdvShow`. Is the image is lined up vertically?
3. If not, repeat the procedure, adjusting the `width` argument until you see vertically aligned data.

If you cannot get a value that gives you vertically aligned data, verify that the number of taps is set properly for your camera.

After you find the correct `width` value, use the same procedure to adjust the `height` value until it is one less than that which results in a timeout from, for example, the `take` application.

Directives

This section describes all of the camera configuration file directives for all cameras supported by EDT. However, for most cameras, only a few of these directives will be needed.

aperture_max

Maximum allowable aperture setting for this camera model. Applies only to cameras with serial aperture control, and only when `serial_aperture` is set.

For example:

```
aperture_max: 18
```

aperture_min

Minimum allowable aperture setting for this camera model. Applies only to cameras with serial aperture control, and only when `serial_aperture` is set.

For example:

```
aperture_min: 0
```

byteswap

Non-Camera Link products only. Sets the BSWAP bit in the Utility2 register; tells the PCI DV whether or not to swap data bytes when transferring the image to the host's memory. A value of 1 swaps bytes; a value of 0 leaves them as is.

By default, `initcam` checks whether the host is little-endian (such as a Sun computer) or big-endian (such as an Intel-based computer) and sets `byteswap` accordingly. However, certain combinations of camera and platform require the use of this directive to override this default behavior.

For example:

```
byteswap: 1
```

camera_class

Required. Argument is a double-quoted string that describes the camera manufacturer. Used with `camera_info` and `camera_model`, which together determine the text displayed in the camera configuration selection dialog to describe a configuration choice.

For example:

```
camera_class: "Redlake"
```

camera_command_file

Specifies the name of a file from which to retrieve setup commands to send to a camera. For example:

```
camera_command_file: camera_config/si600-142_0.txt
```

See `camera_config/si*.txt` files for format and usage. See also [method_camera_download](#).

camera_download_file

Specifies the name of a file from which to retrieve binary camera download data to send to a camera. Binary download files are created by the camera manufacturer and are camera-specific. Download files for some Spectral Instruments cameras are included in the package; however, check with the camera manufacturer to verify that you have a correct and up-to-date download file.

For example:

```
method_camera_download: SPECINST_SERIAL
camera_download_file: camera_config/2350EDT.BIN
```

See also [method_camera_download](#).

camera_info

Required. Argument is a double-quoted string that describes the camera information. Specify at least the minimal information for the camera, model, and operating mode that is unique to this particular configuration file (for example, trigger mode, number of bits, or binning). With [camera_class](#) and [camera_model](#), the information supplied determines the text displayed in the camera configuration selection dialog to describe a configuration choice.

For example:

```
camera_info: "10-bit (monochrome mode)"
```

camera_model

Required. Argument is a double-quoted string that describes the camera model. With [camera_class](#) and [camera_info](#), the information supplied determines the text displayed in the camera configuration selection dialog to describe a configuration choice.

For example:

```
camera_model: "1310C"
```

cameralink

Obsolete. Has no effect.

cameratest

For diagnostic use only. Flag to tell `initcam` to abort immediately after loading the FPGA. For example:

```
cameratest:1
```

cameratyp

Obsolete. Instead use [camera_class](#), [camera_model](#), [camera_info](#).

CL_CFG_NORM

Required for Camera Link cameras and boards. Initial value for register 0x29 Camera Link Control, specified as a two-digit hexadecimal value. Bits are defined as shown below.

Bit	Value	Description
7	0x80	Revision 36 or later of <code>pdvcamlk</code> firmware: Enable region-of-interest padding. As of that revision, if the width or height of incoming data comes up short due to data loss, the region-of-interest logic no longer pads with extra bytes. Set this bit to re-enable the previous functionality. To enable region-of-interest for width only (useful for linescan cameras), set bit 3 (0x08) and bit 7 (0x80). NOTE – Setting this bit on (or using older firmware) can mask timeouts because lost data is padded before the image reaches the device driver, resulting in a persistent out-of-synch condition. The application will then not be notified when it needs to perform timeout recovery.
6	0x40	Set to swap red (R) and blue (B) bytes in RGB triplets, so that they become BGR.
5	0x20	Set to invert the data-valid signal, for the few cameras that require this.
4	0x10	For linescan cameras; enables internal generation of Frame Valid after VACTV lines. Set bit 2 when using this feature.

3	0x08	Set to disable the region-of-interest counters, thus always acquiring the entire image. (The region-of-interest counters are set using the <code>hskip</code> , <code>hactv</code> , <code>vskip</code> , and <code>vactv</code> directives.)
2	0x04	Set to replace the frame-valid signal from the camera with a copy of the line-valid signal instead.
1	0x02	Set if the camera does not implement the data-valid output signal.
0	0x01	Set if the camera is 24-bit color: 8 bits each per red, green, and blue taps. If this bit is set, it overrides any setting of the <code>CL_DATA_PATH_NORM</code> directive.

For example:

```
CL_CFG_NORM: 02 #data valid invert bit set
```

Bits can be OR'd together – for example:

```
CL_CFG_NORM: 16 # ignore data valid, set linescan and
enable VACTV lines feature
```

For details, consult the EDT addendum for Camera Link firmware (see [Related Resources on page 2](#)).

CL_CFG2_NORM

Initial value for register 0x35 Camera Link Control 2, specified as a two-digit hexadecimal value.

Register 0x35 is used differently for different products. For details, consult the appropriate EDT addendum for firmware and the EDT user's guide for framegrabbers (see [Related Resources on page 2](#)).

CL_DATA_PATH_NORM

Required with Camera Link boards and cameras. Initial value for 0x28 Camera Link Data Path, specified as a two-digit hexadecimal value. Bits are defined as shown below.

Bit	Value	Description
7-4		(number of taps) – 1
3-0		(number of bits per pixel) – 1

For example:

```
CL_DATA_PATH_NORM: 19 # dual tap, 10 bits per pixel
```

NOTE Cameras that output color-encoded pixels (typically 8 bits each of red, green, and blue per pixel), instead of Bayer-filtered output, are a special case, and the board instead uses the `CL_CFG_NORM` RGB bit to set the board into 24-bit RGB input mode. In that case, the `CL_DATA_PATH_NORM` directive is ignored; set it to 00 to indicate that it isn't being used.

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

continuous

By default, 0; however, certain cameras need this directive set to 1 when it has been determined that the interframe gap is too brief to wait for a frame-valid.

When 0, the DMA engine waits for a frame-valid signal to determine when to begin transferring pixels from the camera. When 1, it behaves in this way for the first frame only. For subsequent frames, the DMA engine counts pixels until it has transferred `width * height` pixels; then it begins the next frame on the next pixel without waiting for a frame-valid signal.

For example:

```
continuous:1
```

NOTE Set this directive with caution. We have set this value to 1 in configuration files for cameras that need it, in our experience — a small minority of cameras and operating modes. A spurious setting of one can cause you to lose data, resulting in misaligned frames until acquisition ends.

dbl_trig

Non-Camera Link only. An argument of 1 sets the PULNIX bit in the Utility 2 register, which enables Pulnix double-trigger mode. Necessary for board-triggering of certain Pulnix cameras.

For example:

```
dbl_trig: 1
```

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

default_gain

Defines an initial gain value, which `pdv_open` then sets. Not generally useful; applies only to a subset of cameras for which this directive has been set in the camera's default configuration file, as supplied by EDT.

For example:

```
default_gain: 6
```

default_offset

Defines an initial black level. If set, the first call to `pdv_open` then calls `pdv_set_blacklevel` with the specified value. Not generally useful; applies only to a subset of cameras for which this directive has been set in the camera's default configuration file, as supplied by EDT.

For example:

```
default_offset: 100
```

default_shutter_speed

Defines an initial shutter speed, which `pdv_open` then sets. Not generally useful; applies only to a subset of cameras for which this directive has been set in the camera's default configuration file, as supplied by EDT.

For example:

```
default_shutter_speed: 100
```

NOTE If this directive is not set in the configuration file, exposure time remains undefined until set by the application.

depth

Required. Depth of the image, in bits. Must be one of these values: 8, 10, 12, 14, 16, 24, 30, 32. With cameras deeper than eight bits, depth can be set to 8, in which case the board transfers only one byte per pixel (the most significant eight bits).

For example:

```
depth: 8
```

Compare [extdepth](#); see also [mask](#) and [shift](#).

DIRECTION

Obsolete. Ignored.

DIS_SHUTTER

Obsolete.

disable_mdout

Non-Camera Link boards only. A value of 0 clears the ENMCOUTL bit in the Utility register (the default), allowing the four Mode Control signal pairs (MC0–3) to be used for mode control — their usual use.

A value of 1 sets the bit, enabling the PCI DV to use those four signal pairs for incoming data. This is necessary for cameras with pixels of 12 bits or greater that use some of the Mode Control lines for pixel data.

For example:

```
disable_mdout: 1
```

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

DOUBLE_RATE

Non-Camera Link boards only. Enable (1) or disable (0, the default) the clock doubler for high-speed cameras.

For example:

```
DOUBLE_RATE: 1
```

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

Also review bit 0 in the region-of-interest control register.

DUAL_CHANNEL

For dual-channel cameras. A value of 0 (the default) disables dual-channel input; a value of 1 enables it.

For example:

```
DUAL_CHANNEL: 1
```

With Camera Link, or with FOX boards used with the RCX C-Link, use `CL_DATA_PATH_NORM` instead.

ENABLE_DALSA

Non-Camera Link boards only. Enables the EN_DALSA bit in the Configuration Register, which transforms standard AIA exposure control signals to the PRIN / EXSYNC shutter-controlled timing used by some DALSA cameras when running in board-controlled exposure mode.

For example:

```
ENABLE_DALSA: 1
```

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

exposure_max

Alias for `shutter_speed_max`.

exposure_min

Alias for `shutter_speed_min`.

extdepth

Required. The actual number of bits per pixel output by the camera, regardless of how many the DMA engine actually transfers. Possible values are 8, 10, 12, 14, 16, 24, 30, 32.

For example:

```
depth: 8 # 10-bit camera, send only 8 bits.
extdepth: 10
```

Compare [depth](#). See also [shift](#) and compare [markbin](#).

fieldid_trig

Compare [photo_trig](#). Non-Camera Link cameras and boards only. A value of 1 sets the HWTRIG bits of the Utility 2 register, enabling the PCI DV to use the field ID signal pair to trigger the camera from an external source. A value of 0 (the default) clears the bits, which is best for most cameras.

For example:

```
fieldid_trig: 1
```

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

foi_init

Obsolete.

foi_rbtfile

Obsolete.

force_single

Tells the application to use only one buffef, for cases in which the camera uses a serial command or other method to trigger the camera (for example, a Spectral Instruments camera) that violates the ring buffers' pipelining. This flag is used by the API, but if there is a chance that you are using such a camera, then you application must check for this flag as well. To do so, check the return value from `pdv_force_single`. See the EDT API.

NOTE To avoid application failure, if this flag is set, do not start multiple buffers at once, as is the normal, pipelined case (as `take.c` and `simple_take.c` do). For details, see the source code for `take.c`, which checks this flag and deals with the result.

For example:

```
force_single: 1
```

frame_delay

Applies only on a PCI DV or PCI DVK, and only when the `genericssim` directive also is used. The number of lines to delay between frame-valid signals when using board-generated simulator data.

For example:

```
frame_delay: 255
```

frame_height

Used to give an application an idea of the frame height. Not used by the driver or library, this directive has no effect; it is provided only to pass through to applications with GUIs such as `PdvShow`, which can use it, for example, to determine the values that should mark the ends of user-controlled sliders.

For example:

```
frame_height: 80
```

frame_period

Useful for cameras that are not ready for a trigger immediately after acquiring a frame, or when you want the board to output a continuous trigger at a specified interval. Units are microseconds. Applies only when the application specifies `method_frame_timing` with a valid argument — either `FMRATE_ENABLE` or `FVAL_ADJUST`.

Depending on the state of `method_frame_timing`, an integer that sets one of two things:

- if continuous frame triggers are used (`FMRATE_ENABLE`), the interval between triggers;
- otherwise (`FVAL_ADJUST`), the number of microseconds after the end of a frame before starting the next.

See [method_frame_timing](#).

fv_once

Enables (a value of 1) or disables (a value of 0 — the default) continuous acquisition after the first frame valid. Causes the board to acquire successive frames without waiting for subsequent `FVAL` signals to go `TRUE`, after the first. Not normally needed, use this flag when latency between frames is very short, or to overcome the 64 MB per frame limit by capturing single images over multiple buffers. Use with [frame_height](#).

For example:

```
fv_once: 1
```

NOTE This is the same as setting `continuous`, except it has extra logic that prevents timing out on the last frame of a continuous sequence.

fval_done

Camera Link only. Enables (a value of 1) or disables (a value of 0 — the default) image acquisition termination when the frame-valid signal goes `FALSE`. By default, the board terminates acquisition only after the expected image data (`width * height`) is transferred, or the timeout period expires (the subroutine `pdv_get_timeout` is in the EDT API; see [Related Resources on page 2](#)). When `fval_done` is enabled, the board aborts acquisitions if the driver detects the frame valid going `FALSE`, whether or not all of the expected data has come in.

Typically, line scan applications set `height` to the maximum possible number of lines, and the frame valid signal is generated from an external sensor (such as on a conveyor belt). Images are then read into a fixed-sized buffer that may be only partially filled. To determine how many lines were transferred before the frame valid signal terminated the acquisition, use the `pdv_get_lines_xferred` subroutine.

For example:

```
fval_done: 1
```

gain_min

Minimum allowable gain setting for the camera model. Applies only to cameras that have computer-controlled gain, such as Redlake MEGAPLUS serial cameras. Not used by the driver or library, this directive has no effect; it is provided to pass through to GUI applications such as `PdvShow`, which can use it, for example, to determine what values should mark the ends of user-controlled sliders.

For example:

```
gain_min: 0
```

gain_max

Maximum allowable gain setting for the camera model. Applies only to cameras that have computer-controlled gain, such as Redlake MEGAPLUS serial cameras. Not used by the driver or library, this directive is provided to pass through to GUI applications such as `PdvShow`, which can use it, for example, to determine what values should mark the ends of user-controlled sliders.

For example:

```
gain_max: 255
```

genericsim

PCI DV and PCI DVK boards only. If this value is not zero, `initcam` configures the device to simulate a camera; it then generates its own data.

Three nonzero values are possible:

- 1 = *single* `initcam` prompts the user to press **Return** to generate one frame.
- 2 = *continuous* The simulator generates frames continuously.
- 3 = *triggered* The board generates a frame each time an exposure is requested.

For example:

```
genericsim: 3
```

See [sim_height](#), [sim_width](#), and [simulator_speed](#).

For details on Camera Link internal simulator functionality, consult the EDT user's guide for framegrabbers (see [Related Resources on page 2](#)).

hactv

The width, in pixels, of a rectangular region of interest; use with `hskip`, `vskip`, and `vactv` to set the other coordinates. When set, this value overrides the image width throughout the software and firmware.

For example:

```
hactv: 1024
```

See [hskip](#), [vactv](#), and [vskip](#). See also `pdv_set_roi` and `pdv_enable_roi` in the EDT API.

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

Also review the region-of-interest control register.

header_dma

Initial setting for header DMA flag. Causes `initcam` to call `pdv_set_header_dma` with the specified value. Valid arguments are 0 (FALSE—the default) or 1 (TRUE).

For example:

```
header_dma: 1
```

For details on header functionality, review the `pdv_get/set_header_*` subroutines in the EDT API (see [Related Resources on page 2](#)).

header_size

An integer value representing the initial setting for header size. Causes `initcam` to call `pdv_set_header_size` with the indicated value. , valid range 0-65535.

For example:

```
header_size: 1024
```

For details on header functionality, see the `pdv_get/set_header_*` subroutines in the EDT API (see [Related Resources on page 2](#)).

height

Required. Specifies the height, in pixels, of the image data output by the camera. Because some cameras output more lines per image than are in the CCD's active image area, this value does not always match the height described in the camera documentation.

For example:

```
height: 1024
```

If `vactv` is specified, its value overrides that specified in `height`.

See also `vactv` and `vskip`. Compare `width`.

hskip

The upper left X coordinate of a rectangular region of interest; use with `hactv`, `vskip`, and `vactv` to set the other coordinates.

For example:

```
hskip: 10
```

See `hactv`, `vactv`, and `vskip`. See also `pdv_set_roi` and `pdv_enable_roi` in the EDT API.

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

Also review the region-of-interest control register.

htaps

Number of horizontal taps per pixel clock cycle. Sets up the board's DMA logic to sequence the DMA data properly for a correctly displayed image. To display an image correctly, the board must be set correctly for:

- the number of DMA channels, corresponding to the number of taps in the camera (set with `CL_DATA_PATH_NORM`), and
- for two-tap cameras, whether the pixels coming in from alternate taps are supposed to be next to each other on the same line (`htaps: 2`), or in the same relative position on adjacent lines (`vtaps: 2`).

[Figure 1](#) shows the difference between the two types of pixel ordering (for an imaginary camera with only twelve pixels per line). In this figure, pixels are labeled according to the DMA channel, or camera tap, from which they originate.

Figure 1. Horizontal vs. Vertical Pixel Ordering in Two-tap cameras

0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1

htaps: 2 vtaps: 2

For a two-tap camera, set either `htaps` or `vtaps` to 2, but never both, and only if `CL_DATA_PATH_NORM` is set to specify a two-tap camera.

For example:

```
htaps: 2
```

See also [vtaps](#) and [CL_CFG2_NORM](#).

hwpad

Included for backwards compatibility with older PCI DV or PCI DVK boards. For EDT digital video products developed after 1999, which include the region-of-interest functionality, instead clip the image width to a four-byte boundary using the `hactv` directive.

Number of pixels to append to each line of data — useful for cameras that output a number of bytes per line that is not an even multiple of four, setting this value is often necessary for display operability (for example, `PdvShow` and any other applications that use the MFC library). Valid values are 1, 2, or 3.

For example:

```
hwpad: 3
```

image_offset

Obsolete. Use [header_size](#).

interlace

Required for the Photonic Systems CCIR camera only. The offset from the beginning of the frame to the second bank —necessary to properly order the data in memory for correct image display.

For example:

```
interlace: 185101
```

INV_SHUTTER

Invert the polarity on the shutter (EXPOSE) line, so that negative is true. By default, positive is true. Valid values are 0 (positive is true) and 1 (negative is true).

For example:

```
INV_SHUTTER: 1
```

irig_offset

This sets the number of seconds to add to the internal seconds calculation on the framegrabber, to compensate for latencies in transferring the IRIG values into the frame header.

The default value is 2.

irig_raw

If set to 1, the 32-bit IRIG value in the IRIG header represent the BCD values transmitted by the IRIG-B signal. If set to 0, the IRIG value is a 32-bit value representing Unix seconds or seconds since Jan. 1, 1970.

The `Irig2Record` structure defined in `pdv_irig.h` represents the 32 byte header appended to the image stream when the IRIG2 header is enabled. Within that structure is a union representing either the raw BCD or seconds:

```
union {
    u_int seconds;
    ts_raw_t raw;
} t;
```

The `ts_raw_t` struct is defined in `libedt_timing.h`, and is formatted as a 32-bit bitfield:

```
typedefstruct {
    u_long seconds:6;
    u_long minutes:6;
    u_long hours:5;
    u_long days:9;
    u_long years:6;
} ts_raw_t;
```

irig_slave

If set to 1, this assumes that another framegrabber is acting as the IRIG master, and sending the IRIG values over a ribbon cable to the slave boards.

The default value is 0, so a single framegrabber or the master don't need this directive

irris_strip

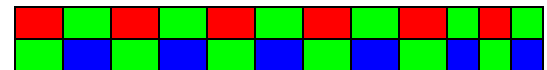
Alias for `hwpad`.

kbs_red_row_first

Only for color cameras that use Bayer filters. A value of 1 indicates the red row is first; a value of 0 indicates the blue row is first.



kbs_red_row_first: 0



kbs_red_row_first: 1

For example:

```
kbs_red_row_first: 0
```

kbs_green_pixel_first

Only for color cameras that use Bayer filters. A value of 1 indicates the green pixel is first in the first row; a value of 0 indicates the green pixel is second in the first row.



kbs_green_pixel_first: 0



kbs_green_pixel_first: 1

For example:

```
kbs_green_pixel_first: 1
```

line_delay

Applies only when the PCI DV or DVK internal simulator is enabled (see `genericsim`). Specifies the number of pixel clock cycles to delay between line valid signals.

For example:

```
line_delay: 255
```

markbin

When enabled (any nonzero value), produces a 32-bit frame counter and replaces four pixels on the output image (for 8-bit per pixel images) or two pixels in the output image (for 10- to 16-bits per pixel images, which display using two bytes per pixel) with the four bytes representing that value. The position of the first pixel

to be replaced is specified by the value given as an argument; this is an offset into the image in host memory. Subsequent pixels are contiguous.

The counter specifies the number of frames that have been acquired since `pdv_open` was last called. Your application can then access this integer by specifying the offset into the image. This can be useful, for example, to ensure that images are being continually acquired when successive images do not change (for example, images from a simulator). Pixels are counted from the top left of the image. Visible effects are minimal; displaying the image shows one pixel flickering.

Because a value of zero (the default) disables the counter, pixel 0, the topmost leftmost pixel, is not available for this purpose.

For example:

```
markbin:4
```

Compare `markras`, `markrx`, `markry`.

markras

When enabled (a value of one), produces a 7-digit frame counter and superimposes it onto an image in host memory in a 56- by 8-pixel rectangle, in the position specified by `markrx`, `markry`. The counter displays the number of frames that have been acquired since `pdv_open` was last called. This can be useful, for example, to ensure that images are being continually acquired when successive images do not change (for example, images from a simulator), or to ensure that no images in a sequence have been deleted. The default value, zero, disables this feature. Use with the directives `markrx` and `markry` to specify where in the frame to display the counter.

For example:

```
markras:1
markry:100
markrx:200
```

Compare `markbin`.

markrx

Assuming that `markras` has been enabled, specifies the X coordinate within the image at which to place the top left corner of the image counter rectangle. Use with the directive `markras` to enable the counter feature, and `markry` to specify the Y coordinate.

For example:

```
markras:1
markry:100
markrx:200
```

Compare `markbin`.

markry

Assuming that `markras` has been enabled, specifies the Y coordinate within the image at which to place the top left corner of the image counter rectangle. Use with the directive `markras` to enable the counter feature, and `markrx` to specify the X coordinate.

For example:

```
markras:1
markry:100
markrx:200
```

Compare [markbin](#).

mask

Non-Camera Link only (for Camera Link, see [CL_DATA_PATH_NORM](#)). The value to which to set the Mask Lo and Mask Hi registers. Bits set to 0 force the corresponding camera data bit to 0.

For example:

```
mask: 3FF (for a 10-bit camera)
```

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

mc4

Obsolete.

method_camera_continuous

Obsolete. Ignored.

method_camera_download

Specifies the download method to use for the file specified by [camera_download_file](#). Valid values are `IRC_160` (some Cincinnati Electronics cameras) and `SPECINST_SERIAL` (Spectral Instruments cameras).

For example:

```
method_camera_download: SPECINST_SERIAL
```

method_camera_shutter_timing

Specifies the use of board-controlled expose timing or one of the camera-specific expose timing methods defined in the EDT API.

NOTE This directive sets only the board's exposure mode, not the camera's. To avoid unpredictable results, make sure the camera is in a compatible exposure mode (typically by means of a serial command).

By default (when no `camera_shutter_timing` directive is present), the board does not control the timing. Instead, timing is controlled by serial commands (sent by the serial command utility `serial_cmd`, or one of the `pdv_serial_command` library subroutines), or by means of Camera Link serial control panel provided by the camera manufacturer, or by some other camera-specific method.

For example, this directive with the argument `AIA_MCL` would be:

```
method_camera_shutter_timing: AIA_MCL
```

Valid arguments are:

<code>AIA_SER</code>	Default. Generic serial timing mode. No timing from the board; camera's exposure time controlled by means of a serial command or other camera-specific method, as is typically the case for freerun mode.
<code>AIA_MCL</code>	Mode control timing. The board holds the CC1 (EXPOSE) line high for the duration of the exposure, which is set using <code>pdv_set_exposure</code> . Units are milliseconds.
<code>AIA_MCL_100US</code>	Mode control timing. Same as <code>AIA_MCL</code> except that the units are microseconds.
<code>COHU_SERIAL</code>	Programmed timing for Cohu 7700 and similar cameras
<code>TOSHIBA_SERIAL</code>	Programmed timing for Toshiba-Teli CS3960CL and similar cameras

PTM6710_SERIAL	Programmed timing for Pulnix TM6710 and similar cameras
PTM1020_SERIAL	Programmed timing for Pulnix TM1020 and similar cameras
TIMC1001_SERIAL	Programmed timing for Texas Instruments TIMC 1001 and similar cameras
ADIMEC_SERIAL	Programmed timing for Adimec 1600, 4020 and similar cameras
BASLER202K_SERIAL	Programmed timing for Basler 202K and similar cameras
SU320_SERIAL	Programmed timing for Sensors Unlimited 320 and similar cameras
SMD_SERIAL	Programmed timing for Dalsa / SMD 1M30, 4M4 and similar cameras

method_flushdma

Obsolete. This functionality is now incorporated in the driver, which flushes DMA before acquiring a frame.

method_frame_timing

Enables the frame timer and determines its function. Valid arguments are:

FMRATE_ENABLE	The board sends continuous triggers to the camera at an interval set by the <code>frame_period</code> directive. (Applies to firmware of version 35 or later.)
FVAL_ADJUST	The board waits until the frame timer value (set by the <code>frame_period</code> directive) has counted down to zero, instead of sending the next trigger immediately upon seeing a frame valid TRUE (as is the normal case). This is necessary for cameras whose minimum interval between frame triggers is longer than the time it takes to acquire and transfer the image.

Both assume `MODE_CNTL_NORM` has been set as appropriate for board triggering. For example:

```
# adjust trigger timing such that a trigger always comes 100 ms after
# beginning of previous frame readout
MODE_CNTL_NORM: 10
method_frame_timing: FVAL_ADJUST
frame_period: 100000

# send a trigger pulse out on EXPOSE line every 300 ms
MODE_CNTL_NORM: 10
method_frame_timing: FMRATE_ENABLE
frame_period: 300000
```

method_header_position

Initial setting for header position, if header is present.

Causes `initcam` to call `pdv_set_header_position` with the indicated value. Valid arguments are `HEADER_BEFORE`, `HEADER_AFTER`, `HEADER_WITHIN`. Typically used along with `header_dma` and `header_size`.

For example, this directive with the argument `HEADER_WITHIN` would be:

```
method_header_position: HEADER_WITHIN
```

Valid arguments are:

```
HEADER_BEFORE
HEADER_BEGIN
HEADER_END
HEADER_AFTER
```

For details about headers, refer to the `pdv_get_header_offset` and `pdv_get/set_header_*` subroutines in the EDT API (see [Related Resources on page 2](#)).

method_header_type

Header methods are used to set up different header (and footer) data, either within the image data, or before or after the image data with extra DMA.

Currently, only one argument is defined for this method: IRIG2. The IRIG2 method is used to tag images with IRIG data (on boards that have that option). Since this method includes frame numbering and other tag data, it can also be used as a way to validate the start of an image and check for missed frames, even on boards that do not have the IRIG option (or IRIG input).

Setting `method_header_type: IRIG2` is equivalent to calling `pdv_set_header_type()` from an application, with a `type = HDR_TYPE_IRIG2`, `irig_slave = 0`, `irig_offset = 2`, and `irig_raw = 0`.

For details on this functionality, see `pdv_set_header_type()` in the EDT API (see [Related Resources on page 2](#)).

method_interlace

Tells the library which method to use to reorder the pixels with a frame, for interleaved or interlaced cameras.

For example, this directive with the argument `BYTE_INTLV` would be:

```
method_interlace: BYTE_INTLV
```

The arguments match up with `#defined` values returned from the library call `pdv_interlace_method`, and the `#defined` values have the prefix `PDV_` as defined in `pdv_dependent.h`.

For example, if the configuration file has this directive / value pair...

```
method_interlace: BGGR
```

...then a subsequent call to `pdv_method_interlace` would return `PDV_BGGR`.

Valid arguments are:

<code>BGGR</code>	Decodes data in Bayer-filtered format for cameras with 8 bits per pixel. Use with <code>kbs_green_pixel_first</code> and <code>kbs_red_row_first</code> to set the decoding to match the specific order of the Bayer filter.
<code>BGGR_WORD</code>	Decodes data in Bayer-filtered format for cameras with 10–16 bits per pixel. Use with <code>kbs_green_pixel_first</code> and <code>kbs_red_row_first</code> to set the decoding to match the specific order of the Bayer filter.
<code>BYTE_INTLV</code>	Bytes are interleaved. Two-byte pairs, from adjacent lines, are contiguous in the data stream. Most commonly used with dual-tap, non-Camera Link cameras.
<code>DALSA_2CH_INTLV</code>	See <code>INVERT_RIGHT_INTLV</code> .
<code>DALSA_4CH_INTLV</code>	See <code>QUADRANT_INTLV</code> .
<code>ES10_BGGR_INTLV</code>	See <code>BGGR</code> .
<code>EVEN_RIGHT_INTLV</code>	Data is organized in pairs of pixels — odd pixels from the left progressing right, and even pixels from the horizontal center, also progressing right.
<code>FIELD_INTLC</code>	Data is organized in pairs of pixels — odd pixels starting at the top left corner of the frame, and even pixels starting at the vertical center, lefthand edge.

<code>INVERT_RIGHT_INTLV</code>	For cameras with sensors like that of the Dalsa A series, which send the data in pixel pairs, odd bytes starting from the left and progressing right, even bytes starting from the right and progressing left, with pixel pairs meeting in the center.
<code>INVERT_RIGHT_BGGR_INTLV</code>	Combined <code>INVERT_RIGHT_INTLV</code> and <code>BGGR</code> . Decodes Bayer-filtered data using the Dalsa A model type sensor.
<code>PIRANHA_4CH_INTLV</code>	See <code>QUADRANT_INTLV</code> .
<code>QUADRANT_INTLV</code>	For cameras with Dalsa four-tap sensors having data coming in in four-pixel quads, progressing inward (first horizontally, then vertically) from the four corners.
<code>SPECINST_4PORT_INTLV</code>	See <code>QUADRANT_INTLV</code> .
<code>WORD_INTLV</code>	Same as <code>BYTE_INTLV</code> but for cameras with over eight bits per pixel.
<code>WORD_INTLV_HILO</code>	Same as <code>BYTE_INTLV</code> except pixels start at the left side of both the top and bottom of the frame, converging toward the center.
<code>WORD_INTLV_ODD</code>	Same as <code>WORD_INTLV</code> except the first pixel comes from the second line.

method_lock_shutter

Obsolete.

method_serial_format

Obsolete. To specify special handling of the `serial_init` string, use one of these special directives: `serial_init_hex`, `serial_init_baslerf`, or `serial_init_duncanf`.

method_serial_mode

Applies only to PCI DVa. Required for RS232 camera serial control through the EDT board. Currently the only valid argument is RS232.

For example:

```
method_serial_mode: RS232
```

To enable RS232 serial control via the EDT board, the RS232 jumper on the board must also be set.

NOTE If this directive is missing, the default is differential – RS422 or LVDS. If the camera uses LVDS or RS422 signals for the serial channel, omit this directive and set the jumper to DIFF.

method_set_gain

Obsolete.

method_set_offset

Obsolete.

method_shutter_speed

An alias for [method_camera_shutter_timing](#).

method_startdma

Obsolete. This functionality is now incorporated in the driver, which flushes DMA before acquiring a frame.

mode16

Enables or disables 16-bit transfers in a FOX board (any version). When OFF (0), 24-bit transfers are used. The mode16 directive applies only when a FOX board is in use; on any other board, it is ignored. Usage is:

```
mode16: 1
```

Setting `mode16` to 1 (on) is the equivalent of configuring a framegrabber-end RCX C-Link to one of the 16-bit modes. By default, it is 0 (off), which is the equivalent of setting the RCX C-Link to a 24-bit mode. The RCX C-Link at the other end must be configured to match the setting of this directive. For configuration codes and details, consult the EDT user's guide for the RCX C-Link (see [Related Resources on page 2](#)).

MODE_CNTL_NORM

Sets the state of the camera control (CC) lines to the camera. Value is an 8-bit hexadecimal number. The right nibble sets the steady state of the four CC lines, and the left nibble selects which of the lines, if any, to use for sending trigger or expose pulses. For cameras in free-run mode, set the left nibble to zero. For cameras that expect a trigger or expose pulse, the left nibble is ordinarily one, because the EXPOSE line for almost all cameras is CC1.

If the leftmost nibble is not zero, the board sends out a one-millisecond pulse on the indicated CC line for every acquire, unless `method_camera_shutter_timing` is set to `AIA_MCL`, in which case the pulse instead lasts for the duration set by the `pdv_set_exposure` library subroutine.

NOTE These modes control the operation of the frame-grabber only. To achieve the expected results, the camera must also be configured to run in a compatible or matching mode, either by means of serial commands (see the directive `serial_init`) or the steady state of another CC line or lines, or some other external means such as a camera manufacturer-supplied application.

For example:

```
# send a trigger or pulse on CC2, set CC2 high (on)
# CC3 and CC4 are low (off)
MODE_CNTL_NORM: 12
```

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

Also review the mode control register and your camera documentation.

offset_min

Minimum allowable offset (black level) setting for the camera model. Applies only to cameras that have computer-controlled offset, such as the Redlake MEGAPLUS serial cameras. Not used by the driver or library, this directive is provided to pass through to GUI applications such as `PdvShow`, which can use it, for example, to determine what values should mark the ends of user-controlled sliders.

For example:

```
offset_min: 0
```

offset_max

Maximum allowable offset (black level) setting for this camera model. Applies only to cameras that have computer-controlled offset, such as Redlake MEGAPLUS serial cameras. Not used by the driver or library, this directive is provided to pass through to GUI applications such as `PdvShow`, which can use it, for example, to determine what values should mark the ends of user-controlled sliders.

For example:

```
offset_max:255
```

pause_for_serial

Can be used to add a pause between serial characters if required for reliable serial communication. Not needed for most cameras. The value is an integer; units are milliseconds.

For example:

```
pause_for_serial: 50
```

pclock_speed

This directive specifies a pixel clock speed, in MHz. It has two uses, as described below.

First, as part of its initial configuration, `initcam` sets an automatic timeout value, based on a relatively slow pixel clock speed of 5 MHz. Therefore, with faster cameras, this directive can be used to bias the automatic timeout in `initcam` to a more appropriate value, resulting in more reasonable wait times for image timeouts in case data loss occurs.

Second, a few non-Camera Link cameras do not generate a pixel clock when the frame-valid signal is false. In this case, use this directive in combination with the `rbtfile` directive, specifying `aia_async.bit` or some other firmware file that uses an asynchronous pixel clock. In such cases, valid values for this directive are 5, 10, and 20.

For example:

```
pclock_speed: 20
```

photo_trig

Compare `fieldid_trig`. A value of 1 sets bit 0 of the Utility 2 register, enabling the PCI DV to use the external optical trigger input pins to trigger the camera. A value of 0 (the default) clears the bit.

For example:

```
photo_trig: 1
```

For the location of these pins, consult the user's guide for EDT framegrabbers.

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

pulnix

Non Camera-Link boards only. Set this to 1 to set the PULNIX bit in the Utility 2 register — required for certain Pulnix cameras, notably the TM-9701 in mode 9.

For example:

```
pulnix: 1
```

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

rbtfile

Required for non-Camera Link boards and the PCI DV FOX. Ignored on Camera Link boards. Specifies which firmware file to download into the onboard FPGA for this device. If no absolute path is specified, searches the `camera_config/bitfiles` subdirectory of the EDT installation package for the appropriate file. The name of the file that gets loaded may vary from the examples shown below.

<code>aiag.bit</code>	PCI DV, PCI DVk, PCI DVa
<code>aiag_2ch.bit</code>	PCI DVa boards with 2-channel cameras of 10 or more bits per pixel
<code>aiagcl.bit</code>	Camera Link and FOX boards (PCI DV C-Link, PCI DV FOX, etc.)

For example:

```
rbtfile: aiagcl.bit
```

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

For specific characteristics of other special firmware files, contact EDT.

rgb30

For 30- to 32-bit cameras, sets the 30-bit RGB multiplex method. Applies only when the board is loaded with 32-bit firmware, such as `pdvcamlk_pir.bit`. Valid values are:

1	Redlake MS and DT series (formerly DuncanTech) 30-bit per pixel cameras that order the data in a manner inconsistent with the Camera Link specification.
3	Camera Link standard cameras.

For example:

```
rgb30: 3
```

For details, consult the EDT application note for Redlake / DuncanTech (see [Related Resources on page 2](#)).

sel_mc4

A value of 1 sets the `SELECT_MC4` bit in the Utility 2 register, enabling the PCI DV to use the signal pair ordinarily assigned to Serial Control Out as a fifth mode control bit. A value of 0 (the default) clears this bit.

For example:

```
sel_mc4: 1
```

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

serial_aperture

For cameras that accept an ASCII serial command with one argument to set the aperture. An ASCII string enclosed in double quotes and passed to the PDV Library subroutine `pdv_set_aperture`, to define the serial command to send when the application calls `pdv_set_aperture`. If the camera uses a different serial format to set the aperture, you can instead set the aperture using calls to lower-level library routines for serial control, such as `pdv_serial_command`; or set up an initial aperture value using `serial_init`.

For example:

```
serial_aperture: "APT"
```

serial_baud

Sets the baud rate (by default, 9600) for cameras that use serial commands for camera control. If this directive is omitted, the default baud rate is used. Valid values are 9600, 19200, 38400, 57600, and 115200.

For example:

```
serial_baud: 115200
```

serial_binit

For cameras that use binary numbers instead of ASCII characters for serial control. A string enclosed in double quotes, representing serial bytes to send out the serial transmit line during camera initialization.

For example:

```
serial_binit: "00 11 22 33"
```

serial_binning

For cameras that accept an ASCII serial command with one argument to set the binning mode. An ASCII string enclosed in double quotes and passed to the PDV Library subroutine `pdv_set_binning`, to define the serial command to send when the application calls `pdv_set_binning`. If the camera uses a different serial format to set the binning, you can instead set the binning mode using calls to lower-level library routines for serial control, such as `pdv_serial_command`; or set up an initial binning mode using `serial_init`.

For example:

```
serial_binning: "BIN"
```

For details on PDV library subroutines, consult the EDT API (see [Related Resources on page 2](#)).

serial_exposure

Enables `pdv_set_exposure` for cameras that accept an ASCII serial command with integer argument to set the exposure time. The argument is a C-language `printf`-style control string which includes zero or one valid `%` integer conversion specifications.

For example:

```
serial_exposure: "EXE %d"
serial_exposure: "SHT %02X"
```

NOTE The ability to parse `%` arguments was added in version 4.2.3.2 of the EDT installation package. If you have an earlier package and your camera's serial protocol is an ASCII command, followed by a space, followed by an integer argument, you can use the legacy format without a `%` conversion specification. In this case, `pdv_set_exposure` will send the argument, followed by a space, followed by the integer value.

For example:

```
serial_exposure: "EXE"
```

Other exposure time methods including trigger pulse timing are available and enabled via `method_camera_shutter_timing`.

serial_gain

Enables `pdv_set_gain()` for cameras that accept an ASCII serial command with integer argument to set the camera's gain. The argument is a C-language `printf`-style control string which includes zero or more valid `%` integer conversion specifications.

For example:

```
serial_gain: "GAE %d"
```

The string can include up to four conversion specifications. For example:

```
serial_exposure: "GAE %02X %02X"
```

In this case, `pdv_set_gain()` will send the same value for all channels so if you use this setting you won't be able to adjust the gains separately on cameras that have multiple sensor channels. We recommend using the direct serial commands such as `pdv_serial_command()` to set the gain in such cases.

NOTE The ability to parse `%` arguments was added in version 4.2.3.2 of the EDT installation package. If you have an earlier package and your camera's serial protocol is an ASCII command, followed by a space, followed by an integer argument, you can use the legacy format without a `%` conversion specification. In this case, `pdv_set_gain` will send the argument, followed by a space, followed by the integer value.

For example:

```
serial_exposure: "GAE"
```

serial_init

For cameras that accept ASCII serial commands, defines a double-quoted, colon-delimited set of serial commands to send the camera at initialization. Use this, for example, to set the camera to a mode compatible with the frame-grabber mode.

For example:

```
serial_init: "RDM 2:TRM N:MDE TR"
```

serial_init_baslerf

Similar to [serial_init](#), except `serial_init_baslerf` causes `initcam` to treat the command string as a series of colon-separated hexadecimal commands and to add Roper/Basler-format framing to each command in the string before sending it to the camera. Specifically, an STX character (0x02) is prepended, and the BCC is calculated and appended along with an ETX (0x03) character. See the PDV Library subroutine `pdv_send_basler_frame` and the Basler A202K user's guide.

For example:

```
serial_init_baslerf: "c00103:a00100:a603530000:a7030b5100"
```

serial_init_duncanf

Similar to `serial_init`, except `serial_init_duncanf` causes `initcam` to treat the command string as a series of colon-separated hexadecimal commands and to add Roper/DuncanTech-format framing to each command in the string before sending it to the camera. Specifically, an STX character (0x02) is prepended, and BCC is calculated and appended. Refer to the DuncanTech User Manual, and the PDV Library subroutine `pdv_send_duncan_frame`.

For example:

```
serial_init_duncanf: "0300160000:04001a01ba00"
```

serial_init_hex

Similar to [serial_init](#), except `serial_init_hex` causes the initialization string to be treated as a series of hexadecimal bytes, separated by spaces, which will be converted from ASCII to binary before being sent to the camera. Use this, for example, to set the camera to a mode compatible with the framegrabber mode. See also the PDV Library subroutine `pdv_serial_binary_command`.

For example:

```
serial_init_hex: "80 82 40 82 80 85 11"
```

serial_offset

Enables `pdv_set_blacklevel` for cameras that accept an ASCII serial command with integer argument to set the camera's black level (offset). The argument is a C-language `printf`-style control string which includes zero or more valid `%` integer conversion specifications.

For example:

```
serial_offset: "BKE %d"
```

The string can include up to four conversion specifications. For example:

```
serial_exposure: "OFS %02X %02X"
```

In this case, `pdv_set_offset()` sends the same value for all channels, so if you use this setting you cannot adjust the offsets separately on cameras that have multiple sensor channels. In such cases, we recommend using the direct serial commands such as `pdv_serial_command()` to set the black level.

NOTE The ability to parse % arguments was added in version 4.2.3.2 of the EDT installation package. If you have an earlier package and your camera's serial protocol is an ASCII command, followed by a space, followed by an integer argument, you can use the legacy format without a % conversion specification. In this case, `pdv_set_gain` will send the argument, followed by a space, followed by the integer value.

For example:

```
serial_exposure: "GAE"
```

serial_response

For ASCII serial commands, sets the expected response from the camera. Used only for a few cameras that have a specific known response to all commands. The application can use this directive to set the expected response to which it can compare the actual response, in order to perform serial command-response handshaking.

For example:

```
serial_response: "Y"
```

serial_term

A string enclosed in double quotes that sets the terminator character(s) to append to ASCII serial commands sent using the `serial_init` directive and other ASCII serial directives, as well as the `pdv_serial_command` library subroutine. The default is `"\r"` (carriage return). To specify no appended characters when sending ASCII serial commands, set this to the empty string (`" "`).

For example:

```
serial_term: ""
```

serial_timeout

The number of milliseconds to wait for a response from a serial command sent by `pdv_serial_wait`. If the camera doesn't respond overtly to serial commands, set this to a value high enough to ensure that the command has time to complete; check the camera manufacturer's specifications for details. Valid values are 0–65535; the default is 1000.

Note that the value set by this directive only effects higher level serial communications such as the `serial_init` phase of the initialization, and convenience routines such as `pdv_set_gain`. When sending serial via the direct serial subroutines such as `pdv_serial_command`, the value set by `serial_timeout` will be ignored since the timeout value gets sent explicitly as part of the subroutine call.

For example:

```
serial_timeout: 500
```

serial_trigger

A string enclosed in double quotes representing the ASCII serial string the driver sends to trigger the camera. Few cameras provide this functionality.

NOTE To avoid timing problems, do not use this directive to trigger the camera when another triggering method is possible.

For example:

```
serial_trigger: "D"
```

serial_waitc

An eight-bit hexadecimal value that sets the expected terminator for serial responses. For ASCII serial cameras, the `pdv_serial_read` subroutine normally waits for the expected number of characters (see library subroutines `pdv_serial_wait` and `pdv_serial_read`) or a serial timeout (see the `serial_timeout` directive) before returning. This directive can help significantly shorten the time it takes for a serial command and response sequence to complete: without it, an application must either know the exact number of characters it expects in response to every serial command, or else wait for the largest possible number of characters, and then wait for the `serial_timeout` value to expire every time `pdv_serial_wait` is called.

For example:

```
serial_waitc: 0D
```

shift

Non-Camera Link boards only. The hexadecimal value to which to set the eight-bit Shift register. Set bit 4 to cause incoming pixels to be sent in the opposite order (most significant bit first); bits 0–3 barrel-shift incoming data by the specified amount. (Bits 5–7 are not used.)

For example:

```
shift: 10
```

For details, consult the appropriate EDT addendum for firmware (see [Related Resources on page 2](#)).

shortswap

Enables (a value of 1) or disables (a value of zero, the default) swapping of shorts (two-byte words). Analogous to `byteswap`. This is a hardware operation and does not degrade performance.

For example:

```
shortswap: 1
```

shutter_speed_frontp

Obsolete.

shutter_speed_min

Minimum allowable shutter speed (exposure time) setting for the camera model. Not used by the driver or library, this setting is provided to pass through to GUI applications such as `PdvShow`, which can use it, for example, to determine what values should mark the ends of user-controlled sliders. When using EDT's camera shutter timer (`MODE_CNTL_NORM 10` and `method_camera_shutter_timing: AIA_MCL`), the minimum is 0; otherwise the setting is camera-dependent (assuming a particular camera's method of shutter timing is implemented in the EDT API).

For example:

```
shutter_speed_min: 0
```

shutter_speed_max

Maximum allowable shutter speed (exposure time) setting for the camera model. Not used by the driver or library, this setting is provided to pass through to GUI applications such as `PdvShow`, which can use it, for example, to determine what values should mark the ends of user-controlled sliders. When using EDT's camera shutter timer (`MODE_CNTL_NORM 10` and `method_camera_shutter_timing: AIA_MCL`), the maximum is 25500, otherwise the setting is camera-dependent (assuming a particular camera's method of shutter timing is implemented in the EDT API).

For example:

```
shutter_speed_max: 20000
```

sim_height

PCI DV and PCI DVK boards only. Height of the image, in pixels. Used by `initcam` to configure the height of the simulated image when the device is in simulator mode.

For example:

```
sim_height: 1024
```

See [genericsim](#). For details on Camera Link internal simulator functionality, consult the user's guide for EDT framegrabbers (see [Related Resources on page 2](#)).

sim_width

PCI DV and PCI DVK boards only. The width of the image, in pixels. Used by `initcam` to configure the width of the simulated image, when the device is in simulator mode.

For example:

```
sim_width: 1024
```

See [genericsim](#). For details on Camera Link internal simulator functionality, consult the user's guide for EDT framegrabbers (see [Related Resources on page 2](#)).

simulator_speed

PCI DV and PCI DVK boards only. Sets the speed at which `genericsim` transfers internally generated data. Applies only when the firmware file is `xtest.bit`. Valid values are:

0	5 MHz
1	10 MHz
2	20 MHz

For example:

```
sim_speed: 2
```

See [genericsim](#). For details on Camera Link internal simulator functionality, consult the user's guide for EDT framegrabbers (see [Related Resources on page 2](#)).

skip

Obsolete.

slop

Obsolete.

timeout_multiplier

Used as a multiplier for the default timeout value. Valid values are 1–65535. Useful with slow cameras when the timeout value that `initcam` (or `pdv_set_exposure`) calculates isn't long enough.

For example:

```
timeout_multiplier: 2
```

Compare [pclock_speed](#) and [user_timeout](#). For details, refer to `edt_timeout` in the EDT API (see [Related Resources on page 2](#)).

TRIG_PULSE

Obsolete.

user_timeout

Sets a specific timeout value, in milliseconds, for acquisition routines to wait for all the image data before returning. Overrides the default, which is to adjust the timeout value automatically depending on exposure time and image size. Normally the automatic value is adequate, and the preferred method for making it longer is to use the `pclock_speed` or `timeout_multiplier` directive. Therefore, use this directive only when a fixed or infinite timeout is required — for example, with some externally triggered camera applications. Valid range is 0 to 65535, with zero indicating forever.

For example:

```
user_timeout: 1000
#wait for one second before timing out
```

vactv

The height, in pixels, of a rectangular region of interest; use with `hskip`, `hactv`, and `vskip` to set the other coordinates. When set, this value overrides the image height throughout the software and firmware.

For example:

```
vactv: 1033
```

See [hactv](#), [hskip](#), and [vskip](#).

For details on region of interest, consult the appropriate EDT addendum for firmware, and `pdv_set_roi` and `pdv_enable_roi` in the EDT API (see [Related Resources on page 2](#)).

variable_size

Obsolete. Ignored.

vskip

The upper left Y coordinate of a rectangular region of interest; use with `hactv`, `hskip`, and `vactv` to set the other coordinates.

For example:

```
vskip: 10
```

See [hactv](#), [hskip](#), and [vactv](#).

For details on region of interest, consult the appropriate EDT addendum for firmware, and `pdv_set_roi` and `pdv_enable_roi` in the EDT API (see [Related Resources on page 2](#)).

vtaps

Number of vertical taps per pixel clock cycle. Sets up the board's DMA logic to sequence the DMA data properly for correct image display, which requires the board to be set correctly for:

- the number of DMA channels, corresponding to the number of taps in the camera (set with `CL_DATA_PATH_NORM`); and
- for two-tap cameras, whether the pixels coming in from alternate taps are supposed to be next to each other on the same line (`htaps: 2`), or in the same relative position on adjacent lines (`vtaps: 2`).

[Figure 1](#) shows the difference between the two types of pixel ordering. For a two-tap camera, set either `htaps` or `vtaps` (but not both) to 2 — only if `CL_DATA_PATH_NORM` is set to specify a two-tap camera. See also [htaps](#) and [CL_CFG2_NORM](#).

For example:

```
vtaps: 2
```

width

Required. Specifies the width, in pixels, of the camera image. Because some cameras output more pixels per line than are in the CCD's active image area, this value does not always match the width described in the camera documentation. If `hactv` is specified, its value overrides that specified in `width`.

For example:

```
width: 1024
```

Compare `height`. See also `hactv` and `hskip`.

xregwrite_xx

Sets specific register values on the board, overriding any values for the register set by other directives.

NOTE Consult EDT before you add or modify any instances of this directive in existing configuration files.

Revision Log

Below is a history of modifications to this guide.

Date	Rev	By	Pp	Detail
20110823	04	PH,SC	21-22	• Added directives (<code>irig_offset</code> , <code>_raw</code> , <code>_slave</code>).
"	04	PH,SC	26	• Under <code>method_header_type</code> , added <code>irig to_offset</code> , <code>_raw</code> , <code>_slave</code> .
"	04	PH	2	• Updated Related Resources slightly.
20110323	03	PH	12-36	• Updated directives.
"	"	"	End	• Added Revision Log
20090000	00-02	LW	All	• Created new guide.