Addendum

Device Configuration Guide

For devices used with EDT framegrabbers

Date: 2015 September 29
Rev.: 0008

International Distributors

Sky Blue Microsystems GmbH
Geisenhausenerstr. 18
81379 Munich, Germany
+49 89 780 2970, info@skyblue.de
www.skyblue.de

In Great Britain:
Zerif Technologies Ltd.
Winnington House, 2 Woodberry Grove
Finchley, London N12 0DR
+44 115 855 7883, info@zerif.co.uk
www.zerif.co.uk

Contact

Sky Blue Microsystems GmbH
Geisenhausenerstr. 18
81379 Munich, Germany
+49 89 780 2970, info@skyblue.de
www.skyblue.de

In Great Britain:
Zerif Technologies Ltd.
H5 Ash Tree Court
Nottingham NG8 6PY, England
+44 115 855 7883, info@zerif.co.uk
www.zerif.co.uk

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...

- **The U.S. Department of Commerce, Export Division, Washington, D.C., U.S.A.**

...or, if the beginning or title page of this documentation shows an ITAR status statement, Buyer should contact...

- **The U.S. Department of Commerce, Export Division, Washington, D.C., 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

Device Configuration Guide

Overview

Before your EDT framegrabber can recognize and access another device, such as 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 (and additionally, serial commands may need to be sent to the camera to put it into the expected mode).

This initialization can be achieved via the configuration files and initialization tools below, all provided in your EDT installation package. In that package, the subdirectory `camera_config` contains initialization / configuration (`.cfg`), files in editable ASCII text, for all cameras tested by EDT.

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

Configuration (.cfg) 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 Configuration (.cfg) File on page 9.

Initialization Tools

After you find or create an appropriate configuration file, you can initialize your EDT board and driver by invoking the `initcam` utility, with an argument specifying the appropriate file. This utility reads the file using the subroutine `pdv_readcfg()` 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.

NOTE Subroutines for EDT digital imaging products, typically starting with `pdv_`, are located in the EDT API (see Related Resources on page 8).

Alternatively, you can perform initialization using tools other than `initcam` – for example...

- Use an EDT capture and display program (`vlviewer` or `pdvshow`). Each has a graphical user interface (GUI). When you first run one of these programs, a Camera Setup dialog first asks you to select your camera configuration; it then creates a script that runs `initcam` to invoke the appropriate configuration file.
- Use 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 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 edt.com and look in the product documentation.

EDT Resources

• Application programming interface (API)	edt.com/api/
• Installation packages (Windows, Linux, etc.)	edt.com/download-hub/
• User guides and datasheets / specifications	edt.com/download-hub/
• Firmware reference – Camera Link	edt.com/downloads/dv-c-link-firmware-reference.pdf
• Pin assignments – Camera Link	edt.com/downloads/cameralinkpinout.pdf

Standards / Specifications

• PCI / PCI Express (PCIe)	www.pcisig.com
• Camera Link	www.visiononline.org
• IRIG-B	irigb.com

Creating a Configuration (.cfg) File

If there is no configuration (`.cfg`) file for your setup, you can copy an existing EDT file, edit it for your needs, rename and resave it, and use it in the same way you would have used the original EDT file.

To create a configuration file for your camera setup:

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`, `generic14-cl.cfg, generic16cl.cfg`
   - Camera Link (RGB color): `generic24cl.cfg`
   - Camera Link (Bayer color): `generic24bcl.cfg`
   - Pre-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 in a text editor (for example, WordPad, eMacs, or VI).

NOTE If you use Notepad, you may have trouble viewing or editing the provided files because Notepad has trouble displaying Unix-style text files without CR/LF line terminators; if so, try using 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 directives related to image height, width, and depth 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 directives for timing and data order to match the camera's output format.
   - For Camera Link, these directives are the following: `CL_DATA_PATH_NORM` and `CL_CFG_NORM`.
   - For pre-Camera Link, these directives are some or all of the following: `shift`, `mask`, `byte`, `shortswap`, `DUAL_CHANNEL`, and `DOUBLE_RATE`.
6. If necessary, add the data reordering directive `method_interlace`.
7. If necessary, add directives for region-of-interest (ROI) or hardware padding 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 at startup.
10. Make other edits as needed (see Directives on page 17).

Required Directives

The directives required for your setup will depend on which camera and which EDT framegrabber you are using. Table 1 summarizes the required directives; for details and for other (optional) directives, see Directives on page 17.

Table 1. Required directives

Required for	Directives	Descriptions
All setups	`camera_class`	Typically specifies the camera manufacturer (e.g., "Acme").
	`camera_model`	Typically specifies the camera model (e.g., "TurboCam 6000").
	`camera_info`	Typically specifies the operating mode (e.g., "1024 x 768 2-tap 8-bit, freerun").
	`width` `height`	The `width` and `height` are the width and height of the camera output. If your camera outputs more pixels per line (for width) or lines per image (for height) than are present in the CCD's active image area, then the values for height and width may differ slightly from the values specified in your camera documentation (because the active pixel area of the CCD is greater than specified. In such cases, you may need to find the correct values for `width` and `height` by trial and error.
	`depth` `extdepth`	The `depth` and `extdepth` are the depth and extended depth of the camera output. The `extdepth` should be set to match the number of bits per pixel in the camera's actual output. The `depth` should be set to match `extdepth` except in these cases: • For 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. • For a 24-bit RGB color camera (8 bits each per R, G, B element), set `depth` and `extdepth` to 24. • For a 30-bit RGB color camera (10 bits each per R, G, B element), set `depth` and `extdepth` to 32. • For a Bayer-filtered camera, set `depth` to 24, and `extdepth` to the number of bits per R, G, B element (usually 8, 10 or 12). If you have issues with `width, height, depth,` or `extdepth,` remember: • Diagonal image skew usually indicates a problem with `width`. • Timeouts or overruns (reported in `vlviewer` or `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 see what a free-running camera is outputting, use `setdebug` for most setups, or `checkcam` for certain legacy setups.
Camera Link setups	`CL_CFG_NORM`	Sets the Camera Link configuration register for specific camera characteristics.
	`CL_DATA_PATH_NORM`	Sets the Camera Link data path register for the correct number of taps and bits.
Certain setups	`rbtfile`	Specifies which user-interface (UI) FPGA firmware to load for certain boards; ignored by all other boards. Always should be present so that your setup is prepared to support any EDT board. For details, see rbtfile on page 36.

Clipping and Padding

If desired, you can use ROI directives to set the region of interest and 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. Additionally, EDT PCIe boards are 16-byte frame aligned, so the total image size (`width * height * bytes per pixel`) must be a multiple of 16.

If you cannot get your data to align 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 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 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.

If you still have problems, verify that `hactv * vactv` is 16-byte aligned; if not, adjust the `hactv` and `vactv` as needed to make it so.

For details on the above directives, see Directives on page 17.

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]
```

Most multi-tap camera link cameras organize the per-tap data horizontally. If your camera does something else, you may need to override the board's default values by adding an `htaps` or `vtaps` directive – for example...

```
htaps: 1
vtaps: 2 #  per-tap data is organized vertically
```

The directive `CL_CFG_NORM` is used to set various other Camera Link-specific attributes, such as line-scan, 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 17.

If your data is noisy, grainy, or banded, 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 countbits on page 13).

For legacy pre-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 legacy 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`, and allowing the default values to take effect. However, if your data is noisy, grainy, 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 countbits on page 13).

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 8). 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.

Camera Mode Control (CC) Lines

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

A few cameras also use the CC lines to control such things as gain, black level, and binning, employing a variety of schemes. For these and other schemes, see `MODE_CNTL_NORM` and consult your camera documentation.

Decoding Bayer-filtered Images

To enable Bayer decoding in the library so you can produce RGB data from Bayer-filtered data, follow the steps below.

1. Copy a working monochrome camera configuration file and save it with 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
# OR
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 then 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 `vlviewer` or `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 render correctly.

For information on enabling and controlling Bayer decoding and white balance from your application, review the subroutine `pdv_set_full_bayer_parameters()`.

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 will not be a multiple of (lines per frame) * (pixels per line).

Extra data at the end of the frame is not likely to cause a problem. However, extra data at the start 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 add such header directives, simply follow these steps:

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

   ```
   count / bytes_per_frame
   ```

   ...where `count` is defined as (lines per frame) * (pixels per line) * (bytes per pixel).

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 17.

Utilities

Your EDT installation package contains a variety of utilities to initialize the EDT framegrabber for your camera and help you verify that your setup is configured properly and operating correctly. These utilities, described in this section, are:

- `initcam` – to initialize the EDT framegrabber.
- `countbits` – to determine whether the pixel data is properly aligned.
- `setdebug` – to determine the driver and software version, determine the number of clocks per line and lines per frame on Camera Link devices, and to view registers and other debugging information.
- `checkcam` – on pre-Camera Link devices only, to determine the number of pixels per line and lines per frame output from a free-running camera.

initcam

To enable a newly created configuration file, you must run `initcam` with the new file, or invoke `vlviewer` or `pdvshow` (since they do not run `initcam` each time they start) use the Camera Setup dialog. Then you can use such applications as `take` or `vlviewer` or `pdvshow`, or an application of your own, to check your results.

For example...

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

When you use the configuration dialog in `vlviewer` or `pdvshow` to select a configuration, the driver creates the file `camsetup0_0.bat`. This file is a script that runs `initcam` with your selected configuration file as an argument.

countbits

The `countbits` utility is designed to help you diagnose pixel alignment errors in image data. 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.

If the pixel data is properly aligned, you should expect to see the following results...

- The least significant bits will show frequent value changes, typically indicating random sensor noise or tiny changes in light or color compared with adjacent pixels.
- The more significant bits will show fewer value changes, typically indicating larger shifts in the actual luminance or color value of the image subject.

Therefore, an image with properly aligned pixel data will show a gradient of value changes, ranging from many changes in the least significant bits to fewer changes in the most. If you do not see this descending gradient, you may have a problem in the cabling or configuration. For example, the cable may be wired incorrectly; or your configuration may have an error in the `CL_CFG_NORM` setting (for Camera Link cameras) or in the `shift`, `mask`, `byteswap`, or `shortswap` settings (for pre-Camera Link cameras). Typically such errors are indicated in the display application (e.g., `vlviewer` or `pdvshow`) by snowy or obviously wrong images.

To use `countbits`, follow the steps below.

1. To acquire and save a raw image, run...

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

2. To run `countbits` on a monochrome 8-bit raw image file, run...

   ```
   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.

setdebug

Use the `setdebug` utility to determine the driver and software version, to get a count of clocks per line and lines per frame (with Camera Link devices), and to view registers and other debugging information.

Run `setdebug -v` to see the EDT device driver and library versions.

Run `setdebug -d 0` for a snapshot of the board registers; note especially the hexidecimal LINESPERFRAME and CLOCKSPERLINE counters (present only on Camera Link boards), which represent the actual lines per frame and pixels per line as the board sees them from the camera. For most multi-tap cameras, multiplying the CLOCKSPERLINE value by the number of taps will yield the number of pixels per line. For details, consult `setdebug` in the user's guide for EDT framegrabbers (see Related Resources on page 8).

checkcam

For pre-Camera Link (PCI DV and PCI DVK) boards only, use `checkcam` to find the number of pixels per line and lines per frame output from a free-running camera.

To use `checkcam`, follow the steps below.

1. Put your camera into continuous output / freerun 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 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 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 your results are not similar to those shown above, then the camera is not in continuous output mode or there is a signal problem. To address such issues, ensure that the camera is outputting good frame valid, line valid and pixel clock signals, and that the cable pinout is correct. For details, refer to the AIA specification.

4. In your 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 configuration file with the published pixels per line and lines per frame for the camera.
2. Run `vlviewer` or `pdvshow` (or some similar display application) and then look at the image data to see if it is vertically aligned.
3. If the data is not vertically aligned, repeat the steps, adjusting the `width` argument until it is.

If you cannot get a value that yields 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 the 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

Sets register 0x0F Utility, bit 0 BSWAP; tells the framegrabber 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 configuration selection dialog to describe a configuration choice.

For example...

```
camera_class: "Acme"
```

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: camdfile.txt
```

camera_download_file

Deprecated.

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 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 configuration selection dialog to describe a configuration choice.

For example...

```
camera_model: "TurboCam 6000"
```

cameralink

Deprecated.

cameratest

Deprecated.

cameratype

Deprecated. Instead, use `camera_class`, `camera_model`, `camera_info`.

CL_CFG_NORM

Required for Camera Link boards. Initial value for register 0x29 Camera Link Control (PDV_CL_CFG), specified as a two-digit hexadecimal value. Table 2 describes bits and examples. For details, consult the appropriate EDT firmware addendum (see Related Resources on page 8).

Table 2. CL_CFG_NORM values

Bit	Hex Value	Description
7	80	Set to enable region-of-interest (ROI) padding. For PCIe firmware rev. 14 and later, if the width or height of incoming data is short due to data loss, the ROI logic no longer pads with extra bytes. Set this bit to reenable the padding. To enable ROI for width only (useful for line-scan cameras): Set this bit and bit 3, below. **NOTE –** Setting this bit (or using older firmware) can mask timeouts because lost data is padded before the image reaches the device driver. If the padding exceeds the amount of lost data, a persistent out-of-sync condition may result. The application then will not be notified when it needs to perform timeout recovery. For PCI firmware prior to rev. 36, this feature (ROI padding) was not present.
6	40	Reserved.
5	20	For the few cameras that require this functionality: Set to invert the data-valid signal.
4	10	For line-scan cameras. Enables internal generation of frame valid after VACTV lines. When using this feature, set bit 2 (0x04) below.
3	08	Set to disable the ROI counters so that the entire image always will be acquired. (The ROI counters are set using the directives `hskip`, `hactv`, `vskip`, and `vactv`.) To enable ROI for width only (useful for line-scan cameras): Set this bit and bit 7, above.
2	04	Set to replace the frame-valid signal from the camera with a copy of the line-valid signal instead.
1	02	Set if the camera does not implement the data-valid output signal.
0	01	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 in the `CL_DATA_PATH_NORM` 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
```

CL_CFG2_NORM

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

Table 3 describes the bits in register 0x35 for the following products:

- Legacy PCIe8 DV (no "a") C-Link framegrabber; and
- PCIe4 DV**a** C-Link, PCIe8 DV**a** C-Link, and PCIe8 DV**a** CLS when used as a framegrabber.

For details on bit descriptions and other products, consult the appropriate EDT addendum for your firmware and the EDT user's guide for your framegrabber (see Related Resources on page 8).

Table 3. CL_CFG2_NORM

Bit	Hex Value	Description
7	80	When set, selects frame rate counter from other channel. 0x35 = 80 selects frame rate counter from channel 1 as source. 0x75 = 80 selects frame rate counter from channel 0 as source. Used when synchronized trigger output is desired on both channels using the internally generated trigger pulse.
6	40	When set, enables onboard automatic re-arm for acquisition when FVAL is detected low.
5	20	When set, replaces the first two bytes of the frame with frame counter.
4	10	When set, replaces the first two bytes of the frame with FF00.
3	08	When set, puts pixels per line register in freerun mode.
2	04	When set, selects trigger 1 input to arm board for acquisition. Primarily set when encoder index pulse is desired to trigger the start of acquisition with a line-scan camera.
1	02	Reserved.
0	01	Reserved.

CL_DATA_PATH_NORM

Required with Camera Link. Initial value for register 0x28 Camera Link Data Path, specified as a two-digit hexadecimal value, as shown in Table 4 and Table 5. For details, consult the appropriate EDT firmware reference (see Related Resources on page 8).

Table 4. Register 0x28 – bit values with example

Bit	Value	Description	EXAMPLE
7–4	–	(number of taps) –1	One example might look like this...
3–0	–	(number of bits per pixel) – 1	`CL_DATA_PATH_NORM:   19   # dual tap, 10 bits per pixel`

Table 5. CL_DATA_PATH_NORM values, page 1 of 2

Taps	Bits	Value	Mode	CL 2.0	Notes
1	8	07	Base	*	–
1	10	09	Base	*	–
1	12	0b	Base	*	–
1	14	0d	Base	*	–
1	16	0f	Base	*	–
1	24	27	Base	*	24-bit RGB (3-tap, 8-bit); set `htaps` to 1 in configuration file.
1	30	29	Medium	*	30-bit RGB (3-tap, 10-bit).
1	36	2b	Medium	*	36-bit RGB (3-tap, 12-bit).
2	8	17	Base	*	–
2	10	19	Base	*	–
2	12	1b	Base	*	–
2	14	1d	Medium	–	–

Table 5. CL_DATA_PATH_NORM values, page 2 of 2

Taps	Bits	Value	Mode	CL 2.0	Notes
2	16	1f	Medium	–	–
2	24	57	Medium	–	"Dual-tap" 24-bit RGB (6-tap, 8-bit).
2	24	d7	Medium	–	"Dual-tap" 24-bit BGR (6-tap, 8-bit), swapped R and B.
3	8	27	Base	*	Unsupported; see CL_CFG_NORM for RGB mode.
3	10	29	Medium	*	–
3	12	2b	Medium	*	–
4	8	37	Medium	*	–
4	8	b7	Medium	–	ABDE ports used, as opposed to ABCD ports used for "37" setting.
4	10	39	Medium	*	2 bytes per pixel, 64-bit DMA.
4	12	3b	Medium	*	2 bytes per pixel, 64-bit DMA.
4	10	b9	Medium	–	2 bytes per pixel (packed as the most significant 8 bits of 4 pixels in 4 bytes, followed by 1 byte with the least significant 2 bits of each ordered 0, 1, 2, 3), 40-bit DMA.
4	12	bb	Medium	–	2 bytes per pixel (acked as the most significant 8 bits of 4 pixels in 4 bytes, followed by 1 byte with the least significant 4 bits of pixel 0 and 1, and 1 byte with the least significant 4 bits of pixels 2 and 3), 48-bit DMA.
4	14	3d	Full	–	2 bytes per pixel, 64-bit DMA.
4	16	3f	Full	–	2 bytes per pixel, 64-bit DMA.
4	16	bf	80-bit	–	2 bytes per pixel, 80-bit DMA.
5	16	4f	80-bit	–	–
8	8	77	Full	*	–
8	10	79	Full	*	Packed: taps 0 through 7 contain the 8 most significant bits of each pixel, ordered from 0 to 7; taps 8 through 9 contain 2 least significant bits of each pixel, ordered from 7 to 0. Unpack in postprocessing, or use `method_interlace: INTLV_10BIT_8TAP_PACKED` or `INTLV_10BIT_8TAP_TO_8BIT`.
10	8	97	80-bit	*	–

* Complies with Camera Link 2.0 specification

CL_MGTSPEED_NORM

For DVa FOX boards – as of rev. 03 firmware, the MGT speed for DMA channels 0, 1, 2, and 3 is set as follows...

Value	Description
00	Sets DMA channels 0, 1, 2, and 3 to 1.25 Gb/s.
01	Sets DMA channels 0 and 1 to 2.5 Gb/s, and 2 and 3 to 1.25 Gb/s.
02	Sets DMA channels 0 and 1 to 1.25 Gb/s, and 2 and 3 to 2.5 Gb/s.
03	Sets DMA channels 0, 1, 2, and 3 to 2.5 Gb/s.

For example, to set all four DMA channels to 1.25 Gb/s, enter...

```
CL_MGTSPEED_NORM: 00
```

For details, consult the appropriate EDT firmware reference (see Related Resources on page 8).

continuous

By default, 0; however, certain cameras need this directive to be 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, the DMA engine does so for the first frame only and then counts pixels until it has transferred `width * height` pixels, after which it begins the next frame on the next pixel without waiting for a frame-valid signal.

For example...

```
continuous: 1
```

CAUTION Set this directive with caution; setting it unnecessarily can cause data underruns to go undetected, resulting in misaligned frames until acquisition ends. EDT has set it to 1 in a few configuration files for cameras and operating modes that need it, such as those providing an interframe gap which is too brief to wait for a frame-valid. If `continuous` is set to 1, consider using the `method_framesync` directive to enable supplemental frame synchronization check.

NOTE Setting `continuous` is the same as setting `fv_once` except that `fv_once` has extra logic which prevents timing out on the last frame of a `continuous` sequence.

PROGRAMMER'S NOTE: To stop and start acquisition while `continuous` is enabled, the subroutine `pdv_stop_continuous()` subroutine must be called explicitly.

dbl_trig

Pre-Camera Link only. Necessary for board-triggering of certain Pulnix cameras.

NOTE Some Pulnix cameras will send the same frame repeatedly even if not triggered. When triggered, the frame buffer in the camera will capture a new image and send it repeatedly. Setting this bit will disable this feature and prevent the framegrabber from capturing these duplicated frames until a new expose has been sent to the camera.

An argument of 1 sets register 0x10 Utility 2, bit 5, to enable Pulnix double-trigger mode.

For example...

```
dbl_trig: 1
```

For details, consult the appropriate EDT firmware reference (see Related Resources on page 8).

default_gain

Sets the initial gain on the input device. Applies only to cameras for which extended control capabilities have been added to the library (see the source code), or to cameras which have a serial command protocol that has been configured using the `serial_gain` directive. Unless you know that one of the above has been implemented for your camera, it is safest to avoid this directive.

For example...

```
default_gain: 6
```

default_offset

Sets the initial black level on the input device. Applies only to cameras for which extended control capabilities have been added to the library (see the source code), or to cameras which have a serial command protocol that has been configured using the `serial_gain` configuration directive. Unless you know that one of the above has been implemented for your camera, it is safest to avoid this directive.

For example...

```
default_offset: 100
```

default_shutter_speed

Sets the initial shutter speed on the input device. Applies only to cameras for which extended control capabilities have been added to the library (see the source code), or to cameras which have a serial command protocol that has been configured using the `serial_gain` configuration directive. Unless you know that one of the above has been implemented for your camera, it is safest to avoid this directive.

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 `CL_DATA_PATH_NORM`.

DIRECTION

Deprecated.

DIS_SHUTTER

Deprecated.

disable_mdout

On Camera Link boards, this directive should not be used, as doing so will result in undefined behavior.

On pre-Camera Link boards, this directive will clear or set register 0x0F Utility, bit 4 ENMCOUTL, to control the four mode control signal pairs (MC0–3), as follows:

- A value of 0, the default, clears the bit, enabling the four signal pairs to be used for mode control — their usual use.
- A value of 1 sets the bit , enabling the four signal pairs to be used for incoming data — 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 firmware reference (see Related Resources on page 8).

DOUBLE_RATE

Pre-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 firmware reference (see Related Resources on page 8).

DUAL_CHANNEL

For pre-Camera Link, 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

Pre-Camera Link boards only. Enables the register 0x02 Configuration, bit 6 EN_DALSA, 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 firmware reference (see Related Resources on page 8).

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`. For Bayer color cameras, set this value to the number of bits per R, G or B element (e.g. 8 or 10, not 24 or 30). See also `shift` and compare `markbin`.

fieldid_trig

Pre-Camera Link boards only. Compare `photo_trig`. A value of 1 sets register 0x10 Utility 2, bits 2-0 HWTRIG, enabling the pre-Camera Link 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 firmware reference (see Related Resources on page 8).

foi_init

Deprecated.

foi_rbtfile

Deprecated.

force_single

Tells the application to use only one buffer, for cases in which the camera uses a serial command or other method to trigger the camera (necessary only in a few rare cases) 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 your application must check for this flag as well. To do so, check the return value from the subroutine `pdv_force_single()`.

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 pre-Camera Link PCI DV or PCI DVK board, and only when the `genericsim` 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

Deprecated.

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 the system (driver) to set the start bit in the DMA write register after the first frame. 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
```

CAUTION Set this directive with caution; setting it unnecessarily can cause data underruns to go undetected, resulting in misaligned frames until acquisition ends. EDT has set it to 1 in a few configuration files for cameras and operating modes that need it, such as those providing an interframe gap which is too brief to wait for a frame-valid. If `fv_once` is set to 1, consider using the `method_framesync` directive to enable supplemental frame synchronization check.

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

PROGRAMMER'S NOTE: To stop and start acquisition while `continuous` is enabled, the subroutine `pdv_stop_continuous()` subroutine must be called explicitly.

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 – see the subroutine `pdv_get_timeout()`. 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 subroutine `pdv_get_lines_xferred()`.

For example...

```
fval_done: 1
```

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. his directive has no effect on driver or library operations other than to provide a pass-through value to applications via the subroutine `pdv_get_max_gain()`. The directive `gain_max` is provided to pass through to a GUI application (e.g., `vlviewer` or `pdvshow`) – which can use it, for example, to determine which values should mark the ends of user-controlled sliders.

For example...

```
gain_max: 255
```

gain_min

Minimum allowable gain setting for the camera model. Applies only to cameras that have computer-controlled gain, such as certain serial cameras. This directive has no effect on driver or library operations other than to provide a pass-through value to applications via the subroutine `pdv_get_min_gain()`. The directive `gain_min` is provided to pass through to a GUI application (e.g., `vlviewer` or `pdvshow`) – which can use it, for example, to determine which values should mark the ends of user-controlled sliders.

For example...

```
gain_min: 0
```

genericsim

Pre-Camera Link PCI DV and PCI DVK boards only; for details on Camera Link internal simulator functionality, consult the appropriate EDT user's guide for framegrabbers (see Related Resources on page 8). If this value is not 0, `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`.

hactv

The width, in pixels, of a rectangular region of interest (ROI); use with `hskip`, `vskip`, and `vactv` to set the other coordinates. When set, this value overrides the image width throughout the software and firmware, except in the case of the subroutine `pdv_get_cam_width()` – which will always return the original value set by the `width` directive.

For example...

```
hactv: 1024
```

Compare the directives `hskip`, `vactv`, `vskip` and see subroutines `pdv_set_roi()` and `pdv_enable_roi()`. See also the directive `CL_CFG_NORM`, which has bits to enable, disable, and otherwise control the ROI, and see register 0x1A Horizontal Active (for Camera Link boards) or register 0x17 ROI (for pre-Camera Link boards).

Consult the appropriate EDT firmware reference. For that resource and the API, see Related Resources on page 8.

header_dma

Initial setting for header DMA flag. Causes `initcam` to call the subroutine `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, see the subroutine `pdv_get/set_header_*()`.

header_size

An integer value representing the initial setting for header size. Causes `initcam` to call the subroutine `pdv_set_header_size()` with the indicated value (the valid range is 0–65535).

For example...

```
header_size: 1024
```

For details on header functionality, see the subroutine `pdv_get/set_header_*()`.

height

Required. Specifies the height, in lines, of the image data output by the camera. Because some cameras output more lines per image than are present 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`. Nonetheless, the subroutine `pdv_get_cam_height()` always returns the value set by the `height` directive.

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

hskip

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

For example...

```
hskip: 10
```

Compare the directives `hactv`, `vactv`, `vskip` and see subroutines `pdv_set_roi()` and `pdv_enable_roi()`. See also the directive `CL_CFG_NORM`, which has bits to enable, disable, and otherwise control the ROI, and see register 0x1A Horizontal Active (for Camera Link boards) or register 0x17 ROI (for pre-Camera Link boards).

Consult the appropriate EDT firmware reference. For that resource and the API, see Related Resources on page 8.

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, correlating 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 should be next to each other on the same line (`htaps: 2`), or in the same relative position on adjacent lines (`vtaps: 2`).

Figure 1 on 26 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	1	0	1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0	1	0	1

htaps: 2

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

vtaps: 2

For a two-tap camera, set either `htaps` or `vtaps` (but never both) to 2, 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 pre-Camera Link PCI DV or PCI DVK boards. For EDT vision 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, settng this value is often necessary for display operability (e.g., `vlviewer`, `pdvshow`, or other applications that use the MFC library). Valid values are 1, 2, or 3.

For example...

```
hwpad: 3
```

image_offset

Deprecated. Use `header_size`.

interlace

Deprecated.

INV_SHUTTER

Sets the register 0x02 Configuration, bit 2 INV_SHUTTER to 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

PCIe Camera Link boards only. 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

PCIe Camera Link boards only. 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

PCIe Camera Link boards only. If set to 1, `irig_slave` 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 this directive is not needed by a single framegrabber or by the master.

irris_strip

Pre-Camera Link PCI DV and PCI DVK boards only. Alias for `hwpad`.

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, as shown in Table 6.

For example...

```
kbs_green_pixel_first: 0
```

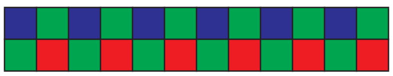
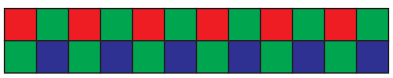
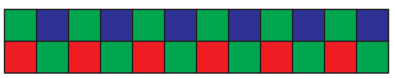
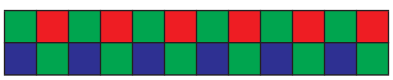
kbs_red_row_first

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, as shown in Table 6.

For example...

```
kbs_red_row_first: 1
```

Table 6. Bayer to RGB directives

Bayer output pattern	kbs_green_pixel_first	kbs_red_row_first
	0	0
	0	1
	1	0
	1	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 the subroutine `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.

Since the default value of 0 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 the subroutine `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 of 0 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

Pre-Camera Link only; for Camera Link, see `CL_DATA_PATH_NORM`. The value to which to set registers 0x12 Mask Lo and 0x13 Mask Hi. 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 firmware reference (see Related Resources on page 8).

mc4

Deprecated.

method_camera_continuous

Deprecated.

method_camera_download

Deprecated.

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 the exposure mode for the board only, not for the camera. To avoid unexpected results, make sure the camera is in a compatible exposure mode (typically via 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 utility `serial_cmd` or one of the `pdv_serial_command()` subroutines – or by the Camera Link serial control panel provided by the camera manufacturer or 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...

`AIA_SER`	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 for freerun mode.
`AIA_MCL`	Mode control timing. The board holds the CC1 (EXPOSE) line high for the duration of the exposure, which is set using the subroutine `pdv_set_exposure()`. The units are milliseconds.
`AIA_MCL_100US`	Mode control timing. Same as `AIA_MCL` except that the units are microseconds.
`COHU_SERIAL`	Programmed timing for Cohu 7700 and similar cameras
`TOSHIBA_SERIAL`	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

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

method_framesync

Enables frame sync header and frame out-of-sync detection.

Framesync is hardware-enabled frame tagging. With framesync enabled, extra header data is added to the frame DMA, including a "magic number" that tags the start of a frame and can be used to check for out-of-sync frames, so that recovery methods can be employed to prevent a persistent out-of-sync state. Valid arguments are:

`FRAMESYNC_OFF`	Framesync functionality disabled.
`FRAMESYNC_ON`	Framesync functionality enabled. Call the subroutine `pdv_check_framesync()` after each acquisition to check whether the image was in sync.
`EMULATE_TIMEOUT`	Framesync functionality enabled; framesync errors will be reflected as timeouts; see the subroutine `pdv_timeouts()`. With this mode enabled, applications that only check for timeouts, including those written before this method existed, can take advantage of this extra synchronization check with no changes to the code.

Framesync functionality is available in PCIe Camera Link framegrabbers except the legacy PCIe4 DV (no "a") C-Link. No PCI devices support this feature.

NOTE For PCI Camera Link boards with firmware rev. 36 and later, and PCIe Camera Link boards with firmware rev 14 and later, the standard timeout (underrun) / overrun logic, as shown in the EDT example applications, usually is sufficient for detecting and recovering from data underruns or overruns. However, with fast cameras that use short blanking intervals, requiring the use of the directives `fv_once` or `continuous`, underruns are not always detected. With such cameras, or with firmware versions earlier than those just mentioned, `method_framesync` can be useful.

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 `frame_period` directive. (Applies to firmware of version 35 or later.)
`FVAL_ADJUST`	The board waits until the frame timer value (set by the `frame_period` 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 (usually CC1) 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 the subroutine `pdv_set_header_position()` with the indicated value. Valid arguments are `HEADER_BEFORE`, `HEADER_AFTER`, `HEADER_WITHIN`. Typically used 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, see the subroutines `pdv_get_header_offset()` and `pdv_get/set_header_*()`.

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). However, the simplest way to do this is to use `method_framesync`.

Setting `method_header_type: IRIG2` is equivalent to calling the subroutine `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 the subroutine `pdv_set_header_type()`.

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: BGGR
```

The arguments match up with #defined values returned from the subroutine `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 subroutine `pdv_method_interlace()` would return `PDV_BGGR`.

Valid arguments are shown in Table 7.

Table 7. Valid arguments for `method_interlace`, page 1 of 2

Bits per pixel	Argument	Description
1	INTLV_1_8_MSB7	1-bit per pixel data, with the most significant bit as the rightmost bit in the output. Outputs 8-bit data converting each bit of input data to 0 or 255 in the output.
1	INTLV_1_8_MSB0	1-bit per pixel data, with the least significant bit as the rightmost bit in the output. Outputs 8-bit data converting each bit of input data to 0 or 255 in the output.
8	BGGR	Decodes data in Bayer-filtered format for cameras with 8 bits per pixel. Use with `kbs_green_pixel_first` and `kbs_red_row_first` to set the decoding to match the specific order of the Bayer filter. The output buffer will be 3 times the size of the input buffer.
8	BGGR_DUAL	Same as `BGGR` but for dual-channel AIA (pre-Camera Link). Legacy.
8	BYTE_INTLV	Adjacent pixels are from even / odd lines.
8	BYTE_INTLV_INOUT	Even pixels start from the center and iterate to the left; odd pixels start from the center +1 and iterate to the right.
8	BYTE_INTLV_MIDTOP_LINE	Even lines start from the center and iterate to the top; odd lines start from the center + 1 and iterate to the bottom.
8	DALSA_LS_4CH_INTLV	4 channel interleave for line-scan sensors, in 4-pixel groups: pix[0], pix[1] iterating to the right and pix[width-2], pix[width-1] iterating to the left, meeting in the center.
8	LINE_INTLV_P3_8X4	4-pixel packets: first, second, third, and fourth vertical quarter (respectively) of the data; inverted horizontally left to right.
8–16	DALSA_2CH_INTLV	Same as `INVERT_RIGHT_INTLV`.
8–16	DALSA_4CH_INTLV	4 channel interleave, adjacent pixels are from line 0, 1, 2, 3, repeating.
8–16	EVEN_RIGHT_INTLV	Data is organized in pairs of pixels. Odd pixels start from the left progressing right; even pixels start from the horizontal center, also progressing right.
8–16	ILLUNIS_INTLV	4-port interleave, ordered lower-right, upper-right, lower left, upper-left, iterating toward the center.
8–16	INVERT_RIGHT_INTLV	Even pixels start from the left edge and iterate to the right; odd pixels start from the right edge and iterate to the left. Even and odd pixels then meet in the center.
8–16	INVERT_RIGHT_BGGR_INTLV	Combined `INVERT_RIGHT_INTLV` and `BGGR`. Decodes Bayer-filtered data using the Dalsa A model type sensor.
8–16	PIRANHA_4CH_INTLV	Same as `QUADRANT_INTLV`.
8–16	QUADRANT_INTLV	4-port interleave, ordered upper left, upper right, lower left, lower right, iterating toward the center.
8–16	QUADRANT2_INTLV	4-port interleave, ordered upper left, upper middle, middle left, middle middle, iterating down and to the right.
8–16	QUADRANT3_INTLV	4-port interleave, upper left, upper right, lower left, lower right, starting with the center 4 pixels, iterating out towards the corners.

Table 7. Valid arguments for `method_interlace`, page 2 of 2

Bits per pixel	Argument	Description
8–24	FIELD_INTLC	Data is organized in pairs of pixels. Odd pixels start at the top left corner of the frame; even pixels start at the vertical center, left edge.
10	INTLV_10BIT_8TAP_PACKED	10-bit data packed into 8 horizontal taps; outputs 10-bit data into 16 bits per pixel buffer with the 6 least significant bits set to 0.
10	INTLV_10BIT_8TAP_TO_8BIT	10-bit data packed into 8 horizontal taps, outputs 8-bit data (strips off LS 2 bits).
10–16	BGGR_WORD	Decodes data in Bayer-filtered format for cameras with 10–16 bits per pixel. Use with `kbs_green_pixel_first` and `kbs_red_row_first` to set the decoding to match the specific order of the Bayer filter.
10–16	SPECINST_4PORT_INTLV	Groups of 4 pixels, upper left, upper right, lower left, lower right, each iterating toward the center.
10–16	WORD_INTLV_MIDTOP_LINE	Even lines start from the center and iterate to the top; odd lines start from the center + 1 and iterate to the bottom.
10–16	WORD_INTLV_TOPMID_LINE	Even lines start from the top and iterate to the center; odd lines start from the center and iterate to the end of the frame.
10–16	WORD_INTLV_HILO_LINE	Even lines start from the top and iterate down; odd lines start from the botttom and iterate up. Even and odd lines then meet in the center.
10–16	WORD_INTLV_HILO	Even lines start from the top and iterate to the center; odd lines start from the center and iterate to the end of the frame.
10–16	WORD_INTLV_TOPBOTTOM	Even lines start from the top and iterate down; odd lines start from the bottom and iterate up. Even and odd lines then meet in the center.
10–16	WORD_INTLV_INOUT	Even pixels start from the center and iterate to the left; odd pixels start from the center +1 and iterate to the right.
10–16	WORD_INTLV	Adjacent pixels are from even / odd lines.
10–16	WORD_INTLV	Same as `BYTE_INTLV` but for cameras with more than 8 bits per pixel.
10–16	WORD_INTLV_HILO	Same as `BYTE_INTLV` except pixels start at the left side of both the top and bottom of the frame, converging toward the center.
10–16	WORD_INTLV_ODD	Same as `WORD_INTLV` except the first pixel comes from the second line.
10–16	WORD_INTLV_ODD	Same as `WORD_INTLV` except it starts with odd lines.
24	BGR_2_RGB	BGR pixels (not Bayer), swap Blue and Green elements.
24	INTLV_24_12	Converts 24-bit packed data to two 12-bit pixels in adjacent 16-bit words. The output buffer will be 1.5 times the size of the input buffer.
24	INVERT_RIGHT_INTLV_24_12	Converts 24-bit packed data to two 12-bit pixels. Channel 0 is the first half of the line (first 12 pixels); channel 0 is the second half (second 12 pixels), reversed. The output buffer will be 1.5 times the size of the input buffer.

method_lock_shutter

Deprecated.

method_serial_format

Deprecated. 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

Pre-Camera Link PCI DVa only. 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 (LVDS or RS422). If the camera uses LVDS or RS422 signals for the serial channel, omit this directive and set the jumper to DIFF.

method_set_gain

Deprecated.

method_set_offset

Deprecated.

method_shutter_speed

An alias for `method_camera_shutter_timing`.

method_startdma

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

mode16

Fiberoptic boards only (e.g., FOX boards, such as PCIe4 DVa FOX); ignored on other boards. Usage is:

```
mode16: 1
```

Setting `mode16` to 1 (on) will set register 0x05 Utility 3, bit 4. Doing so is the equivalent of configuring a framegrabber-end RCX C-Link to one of the 16-bit modes. The default 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 8).

MODE_CNTL_NORM

Sets the state of the camera control (CC) lines to the camera. The 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 typically is set to 1, 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 lines for each acquire, unless `method_camera_shutter_timing` is set to `AIA_MCL`, in which case the pulse instead lasts for the duration set by the subroutine `pdv_set_exposure()`.

To select nonstandard behavior, there are several additional values that can be used in the left nibble:

a - CC1 will be driven high with the trigger signal from the optocoupler (external trigger)

b - CC2 will be driven from the optocoupler; CC1, CC3, CC4 all will be forced low.

c - CC4 will be driven with frame-valid from the camera, for those frames sent to the host.

For details on trigger modes, see the Triggering section of the EDT user's guide for your framegrabber.

NOTE These modes control the framegrabber only. To achieve the expected results, the camera must be configured to run in a compatible or matching mode via serial commands (see `serial_init`), or via the steady state of another CC line or lines, or via some other external means such as an application from your camera manufacturer.

For example...

```
# Typical for free-running cameras:
MODE_CNTL_NORM: 00

# or for cameras set up to expect a trigger or EXPOSE on CC1:
MODE_CNTL_NORM: 10
```

For details, consult the appropriate EDT firmware reference (see Related Resources on page 8).

Also review register 0x07 Mode Control 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. This directive has no effect on driver or library operations other than to provide a pass-through value to applications via the subroutine `pdv_get_min_offset()`. The directive `offset_min` is provided to pass through to a GUI application (e.g., `vlviewer` or `pdvshow`) – which can use it, for example, to determine which 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 a GUI application (e.g., `vlviewer` or `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 pre-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

A value of 1 sets register 0x10 Utility 2, bit 0, enabling the board to use either an external trigger input, or the board-generated fixed period trigger, to trigger the camera and arm the board for acquisition. This value of 1 also disables the driver-serviced end of DMA interrupt, so that the driver will neither re-arm the board nor generate a trigger output.

A value of 0 (the default) clears the bit.

For example...

```
photo_trig: 1
```

For the location of these external trigger (Berg or Lemo connector) pins, consult the user’s guide for EDT framegrabbers.

For details, consult the appropriate EDT firmware reference (see Related Resources on page 8).

pulnix

Pre-Camera Link boards only. Set this to 1 to set the PULNIX bit in 0x10 Utility 2 — required for certain Pulnix cameras, notably the TM-9701 in mode 9.

For example...

```
pulnix: 1
```

For details, consult the appropriate EDT firmware reference (see Related Resources on page 8).

rbtfile

For FPGA configuration, EDT vision boards have two separate register spaces: a user interface (UI) FPGA configuration space, and a PCI / PCIe FPGA configuration space. On some boards – for example, those designed for such older platforms as PCI – the UI FPGA is physically separate from the PCI FPGA, and the UI FPGA configuration file is loaded via `initcam` during initialization. On these boards, the `rbtfile` directive is required in order to specify which FPGA configuration file to load. By contrast, on newer boards – for example, those designed for PCI Express – the single combined FPGA is loaded at boot time via onboard PROM, so the `rbtfile` directive is ignored.

If no absolute path is specified, `initcam` searches the `camera_config/bitfiles` subdirectory of the EDT installation package for the appropriate file. The exact filename may vary from the examples shown below.

`aiag.bit`	For PCI DV, PCI DVK, or PCI DVa boards.
`aiag_2ch.bit`	For PCI DVa boards with two-channel cameras of 10 or more bits per pixel.
`aiagcl.bit`	For PCI DV FOX only; ignored by other Camera Link boards.

For example...

```
rbtfile: aiagcl.bit
```

For details, consult the appropriate EDT firmware reference (see Related Resources on page 8).

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 for 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 for Camera Link standard cameras.

For example...

```
rgb30: 3
```

For details, consult the EDT application note for Redlake / DuncanTech (see Related Resources on page 8).

sel_mc4

Pre-Camera Link boards only. A value of 1 sets register 0x10 Utility 2, bit 4 SELECT_MC4, renabling the framegrabber to use the signal pair ordinarily assigned to Serial Control Out (SCNTLO) 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 firmware reference (see Related Resources on page 8).

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 subroutine `pdv_set_aperture()`, to define the serial command to send when the application calls the subroutine `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

A string enclosed in double quotes, representing serial bytes (in hexadecimal) to send out the serial transmit line during camera initialization. For cameras that use binary numbers instead of ASCII characters for serial control, use this directive (`serial_binit`); for cameras that use ASCII for serial control, use `serial_init` instead.

For example...

```
serial_binit: "00112233"
```

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 subroutine `pdv_set_binning()`, to define the serial command to send when the application calls the subroutine `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, consult the EDT API (see Related Resources on page 8).

serial_exposure

Enables the subroutine `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, the subroutine `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 the subroutine `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, the subroutine `pdv_set_gain()` sends the same value for all channels, so if you use this setting you cannot adjust the gains separately on cameras that have multiple sensor channels. In such cases, we recommend using direct serial command subroutines – for example, `pdv_serial_command()` – to set the gain.

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, the subroutine `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 to the camera at initialization; used, for example, to set the camera mode to one that is compatible with the framegrabber mode. For cameras that accept binary, see `serial_binit`.

For example...

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

Some Basler cameras use a different framing format; see `serial_init_baslerf`.

serial_init_baslerf

For certain older Basler cameras (e.g., 104K, 202K, 402K, 504K, and related series). 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 certain 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 your camera user's guide and the EDT subroutine `pdv_send_basler_frame()`.

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 Geospatial / Goodrich / 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. See your camera documentation and the EDT 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 the EDT subroutine `pdv_serial_binary_command()`.

For example...

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

serial_offset

Enables the subroutine `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, the subroutine `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 mutiple sensor channels. In such cases, we recommend using the direct serial commands – for example, `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, the subroutine `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 subroutine `pdv_serial_command()`. 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 the subroutine `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 – for example, `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 subroutine `pdv_serial_read()` normally waits for the expected number of characters – see subroutines `pdv_serial_wait()` and `pdv_serial_read()` – or a serial timeout (see the directive `serial_timeout`) before returning. This directive can 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 wait for the largest possible number of characters, and then wait for the `serial_timeout` value to expire every time the subroutine `pdv_serial_wait()` is called.

For example...

```
serial_waitc: 0D
```

shift

Pre-Camera Link boards only. The hexadecimal value to which to set the 8-bit register 0x11 Shift. 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 firmware reference (see Related Resources on page 8).

shortswap

Enables (a value of 1) or disables (a value of 0, the default) the swapping of shorts (two-byte words). Analagous to `byteswap`. This is a hardware operation and does not degrade performance.

For example...

```
shortswap: 1
```

shutter_speed_frontp

Deprecated.

shutter_speed_max

Maximum allowable shutter speed (exposure time) setting for the camera model. This directive has no effect on driver or library operations other than to provide a pass-through value to applications via the subroutine `pdv_get_max_shutter()`. The `shutter_speed_max` directive is provided to pass through to to a GUI application (e.g., `vlviewer` or `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
```

shutter_speed_min

Minimum allowable shutter speed (exposure time) setting for the camera model. This directive has no effect on driver or library operations other than to provide a pass-through value to applications via the subroutine `pdv_get_min_shutter()`. The `shutter_speed_min` directive is provided to pass through to to a GUI application (e.g., `vlviewer` or `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 API).

For example...

```
shutter_speed_min: 0
```

sim_height

Pre-Camera Link 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 8).

sim_width

Pre-Camera Link 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 8).

simulator_speed

Pre-Camera Link 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 8).

skip

Deprecated.

slop

Deprecated.

timeout_multiplier

Used as a multiplier for the default timeout value. Valid values are 1–65535. Useful with slow cameras when the timeout value calculated by `initcam` or `pdv_set_exposure()` is too short.

For example...

```
timeout_multiplier: 2
```

Compare `pclock_speed` and `user_timeout`. For details, see the subroutine `edt_timeout()`.

TRIG_PULSE

Deprecated.

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 be when a fixed or infinite timeout is required — for example, with some externally triggered camera applications. Valid range is 0 to 65535, with 0 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 (ROI); use with `hskip`, `hactv`, and `vskip` to set the other coordinates. When set, this value overrides the image height throughout the software and firmware, except in the case of the subroutine `pdv_get_cam_height()` – which always returns the original value set by the `height` directive.

For example...

```
vactv: 1033
```

Compare the directives `hactv`, `hskip`, `vskip` and see subroutines `pdv_set_roi()` and `pdv_enable_roi()`. See also the directive `CL_CFG_NORM`, which has bits to enable, disable, and otherwise control the ROI, and see register 0x1A Horizontal Active (Camera Link boards) or register 0x17 ROI (pre-Camera Link boards).

Consult the appropriate EDT firmware reference. For that resource and the API, see Related Resources on page 8.

variable_size

Deprecated.

vskip

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

For example...

```
vskip: 10
```

Compare the directives `hactv`, `hskip`, `vactv` and see subroutines `pdv_set_roi()` and `pdv_enable_roi()`. See also the directive `CL_CFG_NORM`, which has bits to enable, disable, and otherwise control the ROI, and see register 0x1A Horizontal Active (Camera Link boards) or register 0x17 ROI (pre-Camera Link boards).

Consult the appropriate EDT firmware reference. For that resource and the API, see Related Resources on page 8.

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 on page 26 shows the difference between the two types of pixel ordering.

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

For example...

```
vtaps: 2
```

See also `htaps` and `CL_CFG_NORM`.

width

Required. Specifies the width, in pixels, of the camera image. Because some cameras output more pixels per line than are present 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`. Nonetheless, the subroutine `pdv_get_cam_width()` always returns the value set by this `width` directive.

For example...

`width: 1024`

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

xregwrite_*N*

The *N* fragment of the directive indicates the address of the register in decimal. The argument is the value to set the register, specified as a two-digit hexadecimal value. For example...

`xregwrite_41: 02` # write the value 0x02 to register 41 (hexadecimal address 29)

CAUTION This directive writes registers directly, circumventing the flow of control provided by the standard directives. It should only be used when the required functionality cannot be achieved using other methods.

xregwrite_0x*XX*

The *XX* fragment of the directive indicates the address of the register in hexadecimal. The argument is the value to set the register, specified as a two-digit hexadecimal value. For example...

`xregwrite_0x29: 02` # write the value 0x02 to register 0x29

CAUTION This directive writes registers directly, circumventing the flow of control provided by the standard directives. It should only be used when the required functionality cannot be achieved using other methods.

Revision Log

Below is a history of modifications to this guide.

Date	Rev	By	Pp	Detail
20150929	0008	PH,RH	All	• All pgs: Changed "Camera Configuration Guide" to "Device Configuration Guide."
20150929	0008	PH,RH	43	• Revised one directive, added another, and changed the Caution on both: – Revised `xregwrite_xx` to `xregwrite_N` with this (revised) text: The *N* fragment of the directive indicates the address of the register in decimal. The argument is the value to set the register, specified as a two-digit hexadecimal value. For example... `xregwrite_41: 02` # write the value 0x02 to register 41 (hexadecimal address 29) – Added xregwrite_0x*XX* with this (new) text: The *XX* fragment of the directive indicates the address of the register in hexadecimal. The argument is the value to set the register, specified as a two-digit hexadecimal value. For example... `xregwrite_0x29: 02` # write the value 0x02 to register 0x29 – Revised the Caution on both to say: Caution: This directive writes registers directly, circumventing the flow of control provided by the standard directives. It should only be used when the required functionality cannot be achieved using other methods.
20150903	0007	PH,CH	36	• In Directives, revised photo_trig as follows: "A value of 1 sets register 0x10 Utility 2, bit 0, enabling the board to use either an external trigger input, or the board-generated fixed period trigger, to trigger the camera and arm the board for acquisition. This value of 1 also disables the driver-serviced end of DMA interrupt, so that the driver will neither re-arm the board nor generate a trigger output."
20141121	next	PH,CH	21	• In Directives, added CL_MGTSPEED_NORM.
20140606	0006	PH,RH	Title pg 11-12 18, 21-22 All	• Simplified subtitle to just EDT framegrabbers. • Updated Utilities (initcam, countbits). • Updated Directives (continuous, fv_once). • To text on such display applications as `pdvshow`, added `vlviewer` as appropriate.
20140425	05a	PH,RH	30-31	• To Overview, paragraph 1, added: "...(and additionally, serial commands may need to be sent to the camera to put it into the expected mode)." • To Table 7 (arguments), added QUADRANT2_INTLV and QUADRANT3_INTLV.
20130423	05	PH,RH, CH,DB	Various	Made multiple updates – especially to directives, including but not limited to... • P. 17-20 – CL_CFG_NORM, CL_CFG2_NORM, CL_DATA_PATH_NORM: Revised / expanded details; added tables. • P. 23 – fv_once: Made slight correction to sentence 2 and clarified the note. • P. 24 – hactv, hskip: Updated • P. 30 – method_framesync: Added. • P. 31-32 – method_interlace: Revised / expanded details; added table. • P. 33 – mode16: Updated. • P. 33-34 – MODE_CNTL_NORM: Updated. • P. 41 – vactv, vskip: Updated.
20130423	05	PH	All	• Repaginated to use continuous arabic numerals from title page to end. • Implemented new terminology: Changed "digital video" to "vision" or "digital imaging."
20110823	04	PH,SC	21–22	• Added directives (irig_offset, _raw, _slave).
"	04	PH,SC	26	• Under method_header_type, added irig to _offset, _raw, _slave).

Date	Rev	By	Pp	Detail
"	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.

Contact

Sky Blue Microsystems GmbH
Geisenhausenerstr. 18
81379 Munich, Germany
+49 89 780 2970, info@skyblue.de
www.skyblue.de

In Great Britain:
Zerif Technologies Ltd.
H5 Ash Tree Court
Nottingham NG8 6PY, England
+44 115 855 7883, info@zerif.co.uk
www.zerif.co.uk