

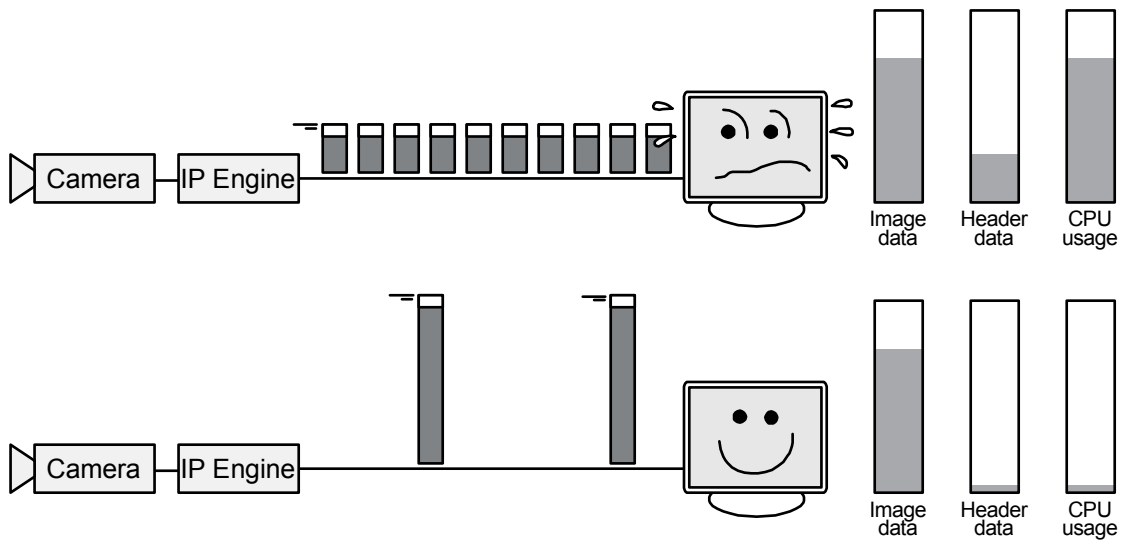


**iPORT**

# Concepts Guide







from "Jumbo packets" on page 15

...proven performance

Version 2.4

These products are not intended for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Pleora Technologies Inc. (Pleora) customers using or selling these products for use in such applications do so at their own risk and agree to indemnify Pleora for any damages resulting from such improper use or sale.

Copyright © 2008 Pleora Technologies Inc. All information provided in this manual is believed to be accurate and reliable. No responsibility is assumed by Pleora for its use. Pleora reserves the right to make changes to this information without notice. Redistribution of this manual in whole or in part, by any means, is prohibited without obtaining prior permission from Pleora.

1/22/08

# Contents

<b>About</b> .....	<b>7</b>
<b>DHCP servers</b> .....	<b>9</b>
<b>IP Engine network detection</b> .....	<b>11</b>
<b>Unique subnets</b> .....	<b>13</b>
<b>Jumbo packets</b> .....	<b>15</b>
<b>Heartbeat function</b> .....	<b>17</b>
<b>Interpacket delay (flow control)</b> .....	<b>19</b>
<b>Grabbing (image acquisition)</b> .....	<b>21</b>
<b>Unicast, multicast, and multi-unicast modes</b> .....	<b>25</b>
<b>Pixel alignment</b> .....	<b>27</b>
<b>Packing</b> .....	<b>29</b>
<b>Binning, decimation, and area of interest</b> .....	<b>33</b>
<b>GigE Vision</b> .....	<b>37</b>
<b>Index</b> .....	<b>41</b>



# About

Acquiring images over Ethernet requires knowledge in many disciplines – you need to have an understanding of cameras, image processing, networking, programming, computer architecture, software, and more. Any one of these disciplines could involve a lifetime of learning – it’s unrealistic to expect one person to be an expert in all of them.

Though Pleora’s iPORT Connectivity Solution reduces the amount of expert knowledge you require, having a basic understanding of *how* things work can be a great asset when things *don’t work* as you expect.

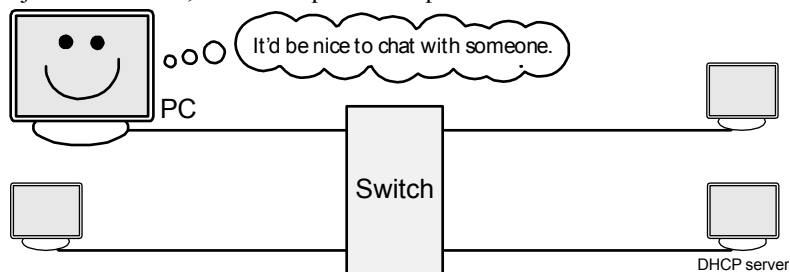
This guide describes the concepts behind much of the functionality for your iPORT Connectivity Solution. Though the concepts are presented in a brief, highly illustrated, easy-to-read (and hopefully fun) form, they include detailed technical information. Many topics include an “In practice” section that explains how you can apply what you’ve learned.

The concepts are roughly ordered, but feel free to browse, read what interests you, and enjoy.

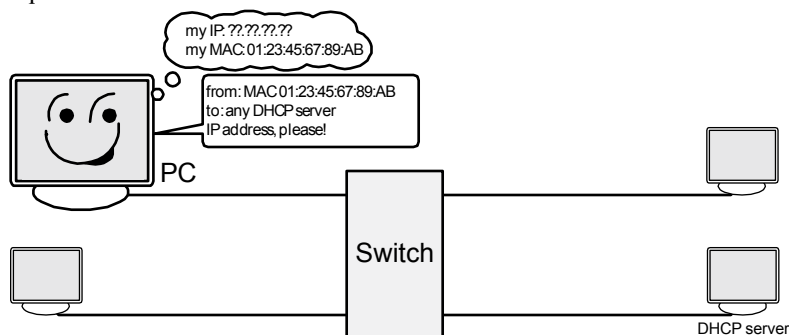


# DHCP servers

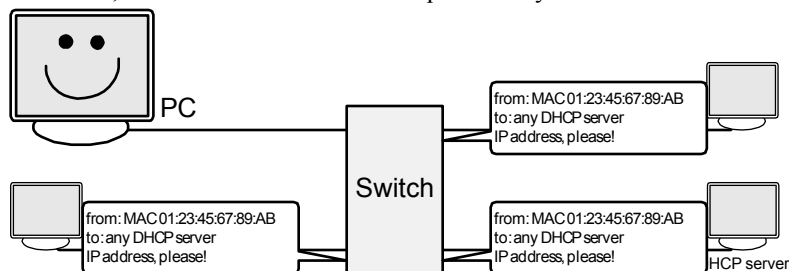
When a PC joins a network, it must acquire a unique IP address.



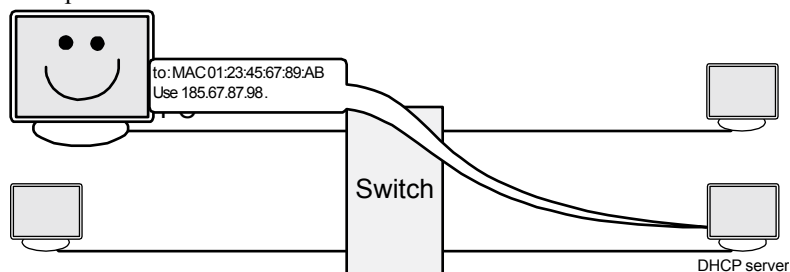
The PC doesn't yet have an IP address, but it does have a unique MAC address. Using layer 2 (MAC layer), the PC requests an IP address from the DHCP server.



To find the DHCP server, the switch broadcasts the request on layer 2.

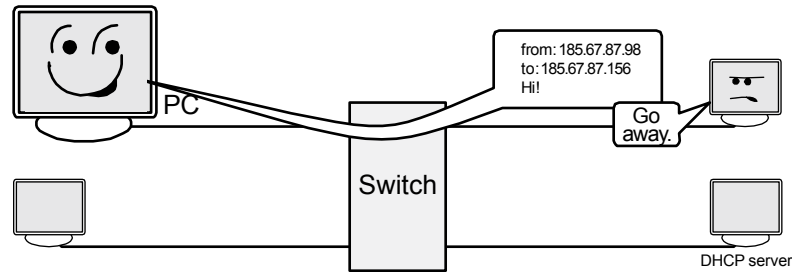


The DHCP server replies with an IP address.



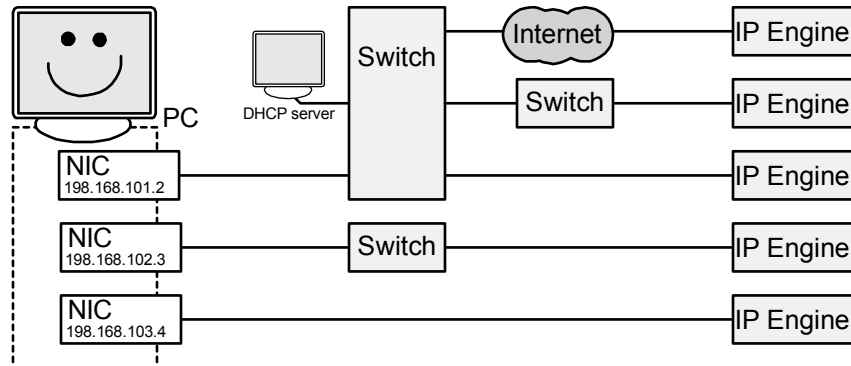
## 10 DHCP servers

Equipped with an IP address, the PC can now communicate with other network appliances using layer 3.

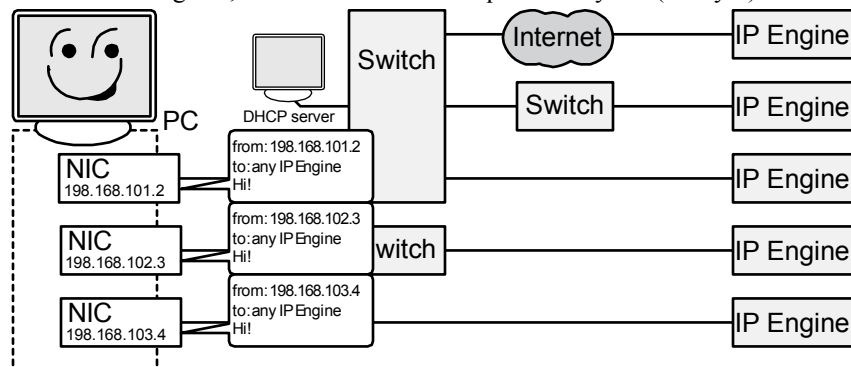


# IP Engine network detection

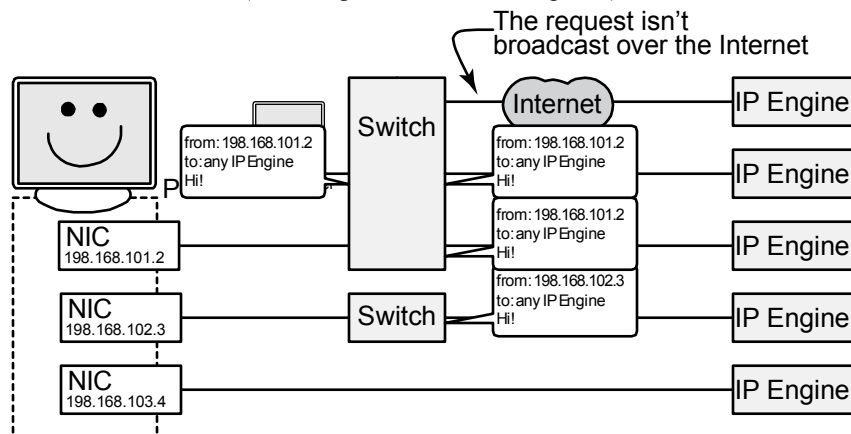
A typical network could have IP Engines connected directly to your NIC, to a switch (or multiple switches), and through the Internet.



To find any available IP Engines, the PC broadcasts a request on layer 3 (IP layer).



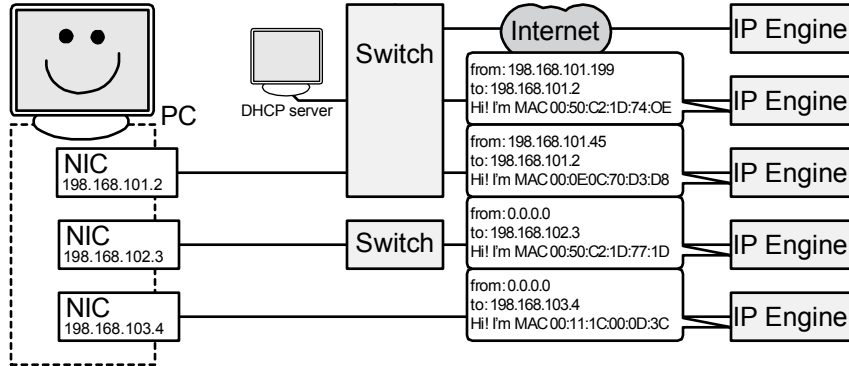
Switches propagate the request. The switches actually broadcast everywhere, but the gateway drops requests to different subnetworks (including Internet-bound requests).



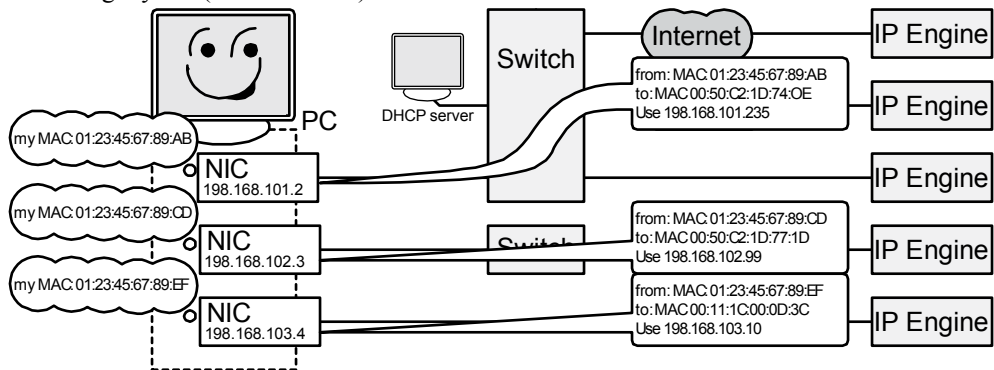
The IP Engines reply on layer 3. If the IP Engine is on a network with a DHCP server, it will automatically acquire an IP address (similar to the PC described in "DHCP servers" on page 9). Any IP Engines

## 12 IP Engine network detection

with a static IP address saved in flash will also already have an IP address. Otherwise, the IP Engines boot with an IP address of 0.0.0.0.



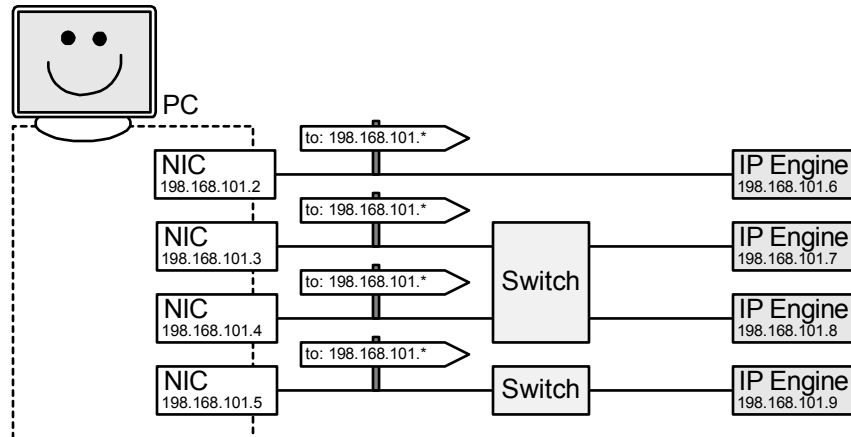
When connecting to an IP Engine, the PC can optionally change the IP address (0.0.0.0 addresses must be changed). Since the PC can't send a layer 3 (IP address) message to an IP address of 0.0.0.0, it sets IP addresses using layer 2 (MAC address).



**NOTE!** IP Engines with firmware versions 4.00 and higher can get their IP address from a DHCP server. IP Engines with earlier firmware versions require a BOOTP (Bootstrap Protocol) server. However, DHCP is based on BOOTP and most DHCP servers also offer BOOTP support.

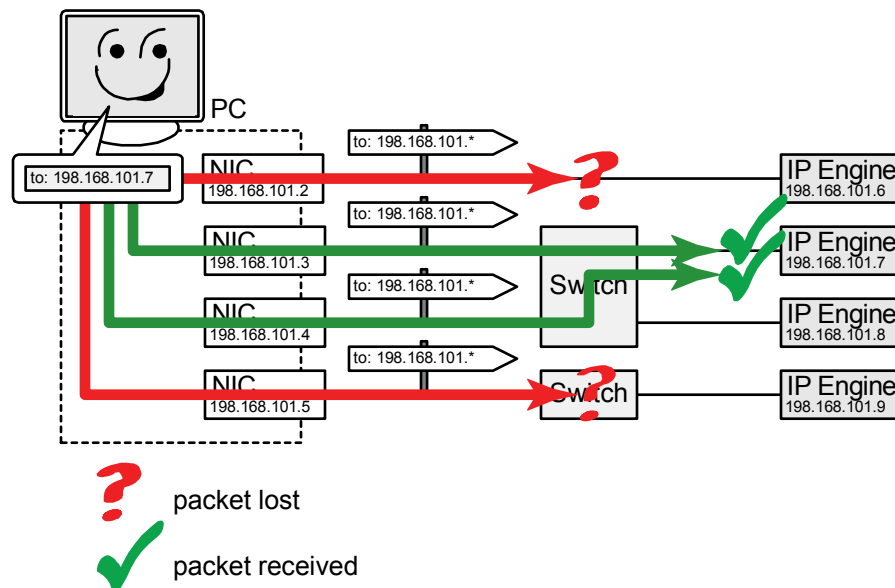
# Unique subnets

When packets can't be sent directly to a receiver, the sender must decide where to send the packet for the next hop. RFC 1122 section 3.3.1 doesn't specify how a switch chooses the next hop, but requires that it choose *only one* path. The switch doesn't know about connections beyond the next hop and if it chooses the wrong path, the packet gets lost.



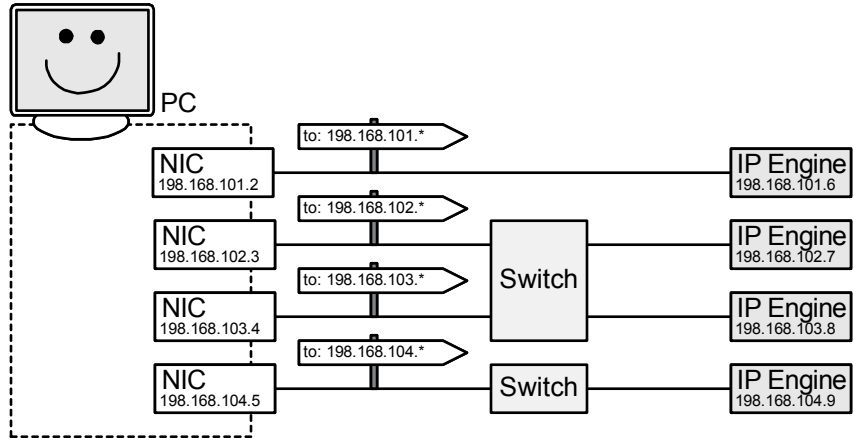
**NOTE!** This example presumes the subnet mask is 255.255.255.0. Thus, 198.168.101.\* and 198.168.102.\* are on different subnets. To learn more about subnets and subnet masks, see <http://en.wikipedia.org/wiki/Subnetwork>.

In this particular case, the packet has a 2 in 4 chance of being successfully received and the connection may appear to work. However, even a successful test doesn't ensure that the arrangement will work indefinitely. If you install a new NIC or change the switch arrangement, the packets may no longer successfully reach their destination.

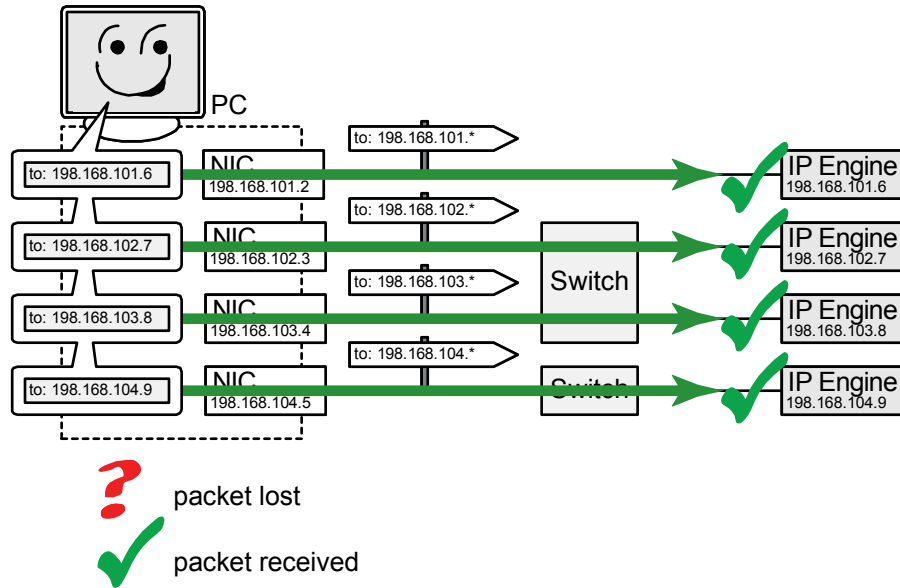


## 14 Unique subnets

By using different subnets for each NIC/IP Engine pair, no guessing is required.



Thus, every packet will consistently and reliably reach its destination.



## In practice...

### To configure your NIC's IP address:

- See the *eBUS Quick Start Guide*.

### To configure your IP Engine's IP address:

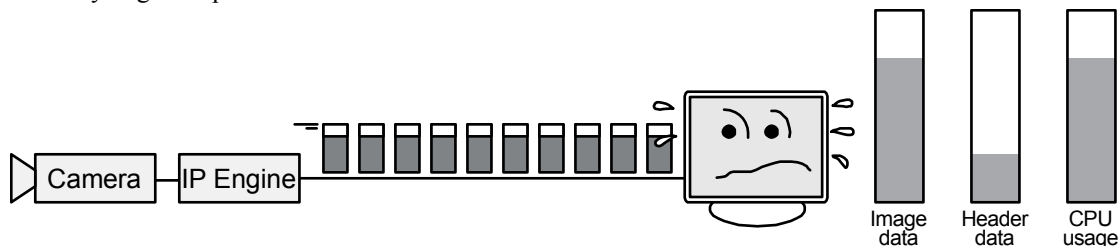
- See the *iPORT Quick Start Guide*.

### To configure IP address values programmatically:

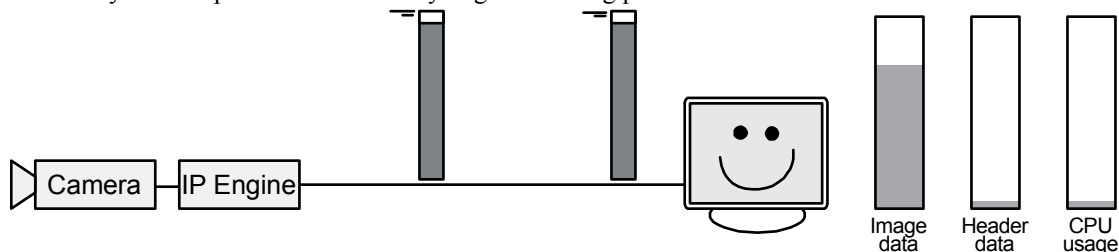
- See the *iPORT C++ SDK Reference Guide* (CY\_DEVICE\_PARAM\_ADAPTER\_ID for your PC's NIC; CyDeviceFinder::SetIP for your IP Engine).

# Jumbo packets

In the early days of networking, the maximum transmission unit (MTU) was 1518 bytes. At the time, communication speed was low and error rates were high. With today's high transmission rates, the task of analyzing each packet can overwhelm the CPU.

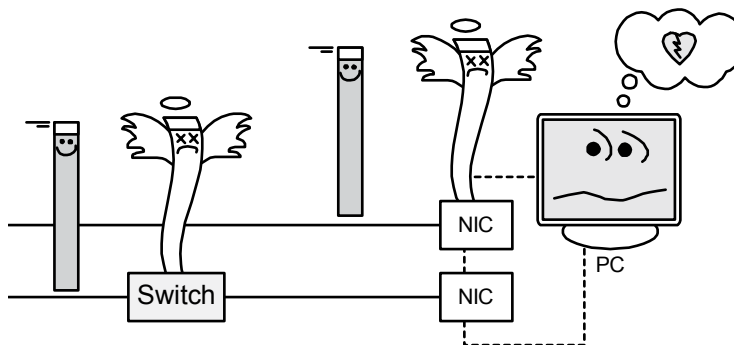


By using jumbo packets, you can transmit the same amount of data with fewer packets. Though you save a small amount of bandwidth (by using fewer headers), you dramatically reduce CPU usage because your PC spends less time analyzing and routing packets.



A common jumbo frame size (or MTU) is 9000 bytes. Though IPv4 supports jumbo packets up to 64 KB, the 32-bit CRC is less effective above 12000 bytes. In general, we recommend a packet size of 9000 bytes (the iPORT protocol uses 8128 bytes of data and up to 72 bytes for headers and footers).

Many switches and NICs don't support jumbo packets by default. If they don't, your jumbo packets are discarded.



Before using jumbo packets, you should ensure that all the switches and NICs between your IP Engine and PC support your preferred packet size.

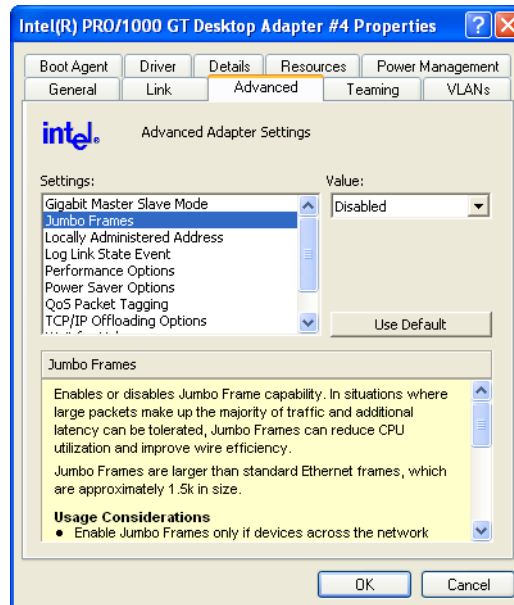
## In practice...

### To configure packet size in Coyote:

- See the *iPORT Coyote Software Guide* (**Options** dialog > **Connection Options** tab).

### To configure your NIC for jumbo packets:

- If you're using an Intel NIC, see the *eBUS Advanced Configuration Reference Guide* (for all other NICs, see your NIC manufacturer's documentation).



### To configure your switch for jumbo packets:

- Consult the switch manufacturer's documentation. Not all switches support jumbo packets.

### To test if jumbo packets will reach your IP Engine:

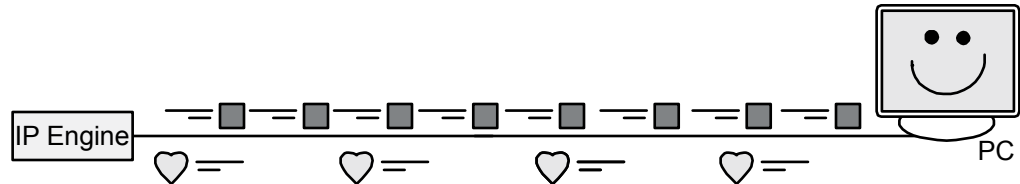
1. Using Coyote, connect to your IP Engine. See the *iPORT Quick Start Guide*.
2. Using Coyote, acquire images with packet sizes of 1440 bytes.  
You should be able to acquire images regardless of the switch and NIC configurations.
3. Change the packet size to 8128 and try acquiring images again.  
If you successfully acquire images, your configurations are good for jumbo packets (up to 9000 bytes). If not, adjust your switch and NIC settings. Your configuration allows jumbo packets if no packets are lost. If packets are lost, you must reconfigure your switch or your NIC, then retest. If you can't configure your system for jumbo packets, divide the packet size by two until you can successfully acquire images.

### To set packet size programmatically:

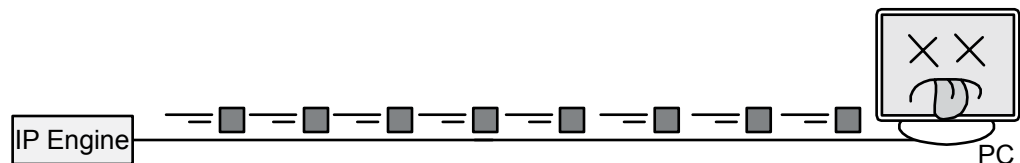
- See the *iPORT C++ SDK Reference Guide* (CY\_CONFIG\_PARAM\_PACKET\_SIZE).

# Heartbeat function

The heartbeat function lets the IP Engine send data only when the host PC is properly connected to the network.



Using the heartbeat function keeps the IP Engine from endlessly transmitting data to a PC that has long since disconnected from the network (due to power outage, network failure, etc.).



If the IP Engine doesn't receive a heartbeat signal before the configured maximum time, it stops transmitting (as if it received a stop request).



To restart transmission, the PC must reconnect to the IP Engine and explicitly start transmission.

Normally, the IP Engine accepts heartbeat signals only from the PC to which it is connected. However in Multicast mode, the IP Engine accepts heartbeats from any PC in the multicasting group.

## In practice...

### To configure heartbeating in Coyote:

- See the *iPORT Coyote Software Guide* (**Options** dialog > **Connection Options** tab).

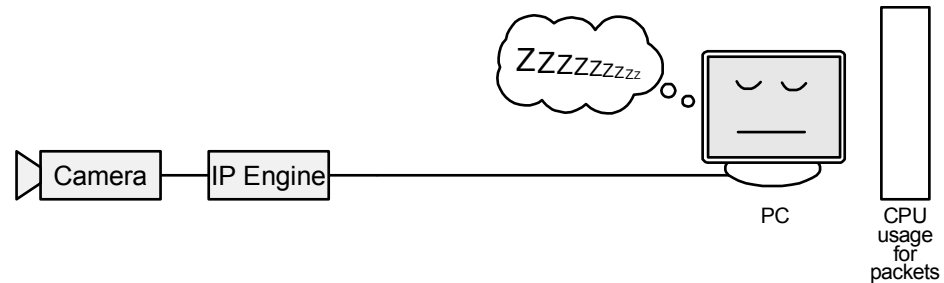
### To configure heartbeating programmatically:

- See the *iPORT C++ SDK Reference Guide* (`CY_CONFIG_PARAM_ENABLE_HEARTBEAT` and other `CY_CONFIG_PARAM_HEARTBEAT` parameters).

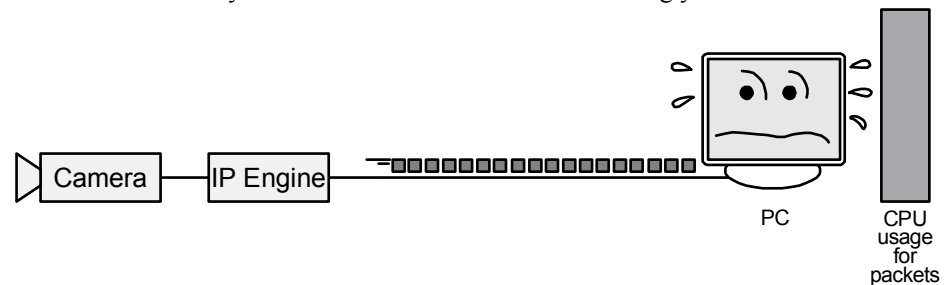
## 18 Heartbeat function

# Interpacket delay (flow control)

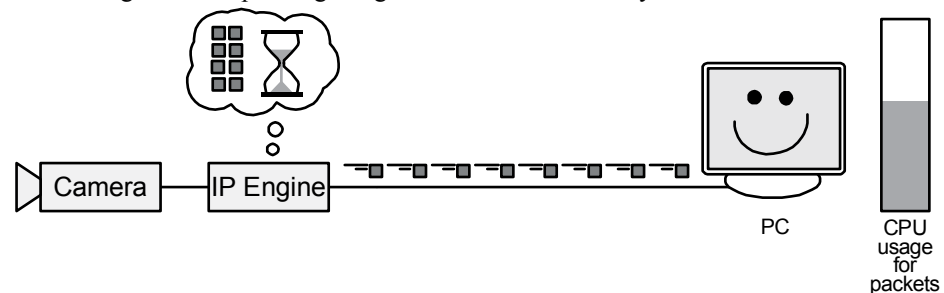
By default, your IP Engine transmits image data to your PC as quickly as possible. However, this can cause long pauses with no activity...



...followed by short data bursts that overwhelm your PC (or a wireless router). Using an eBUS driver or an iPORT driver dramatically reduces the likelihood of overwhelming your PC.



Adding an interpacket delay causes the IP Engine to wait for a moment before sending the next packet (effectively throttling the maximum bandwidth). The delay should be long enough to keep from overwhelming the PC with data bursts, but short enough that image data doesn't form a bottleneck at the IP Engine. The IP Engine stores pending images in its internal memory.



## In practice...

**To minimize the risk (and severity) of overwhelming your PC with packets:**

- Install an eBUS driver or an iPORT driver. See the *eBUS Quick Start Guide*.

**To configure interpacket delay settings in Coyote:**

- See the *iPORT Coyote Software Guide* (**Configuration** dialog > **IP Engine** tab).

## 20 Interpacket delay (flow control)

**To configure interpacket delay programmatically:**

- See the *iPORT C++ SDK Reference Guide* (CY\_DEVICE\_EXT\_FLOW\_CONTROL).

# Grabbing (image acquisition)

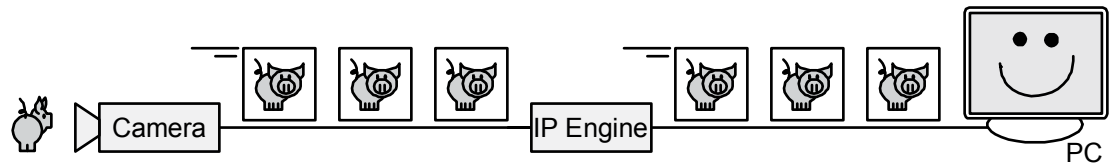
Your iPORT Connectivity Solution lets you configure your system to let the camera, IP Engine, or PC ultimately control when to acquire an image. Understanding your choices lets you better choose the option that suits your needs.

## Method 1: Free run

Camera: Free run

IP Engine: Free run

The camera acquires images as quickly as it can. The IP Engine's grabber acquires every image and sends each on to the PC.

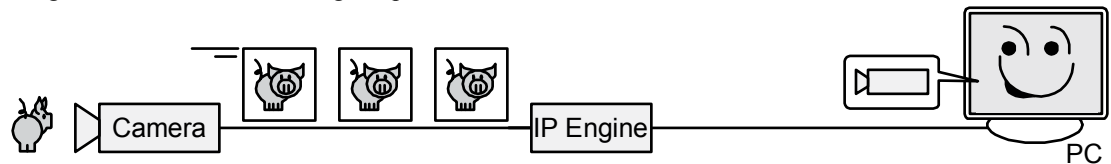


## Method 2: Free run, PC controlled

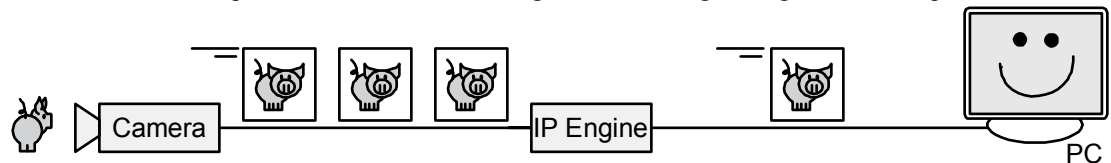
Camera: Free run

IP Engine: Triggered by PC

The camera acquires images as quickly as it can. However, the IP Engine disregards the incoming images until it receives an image request from the PC.



Once it receives a request from the PC, the IP Engine sends a single image for each request.



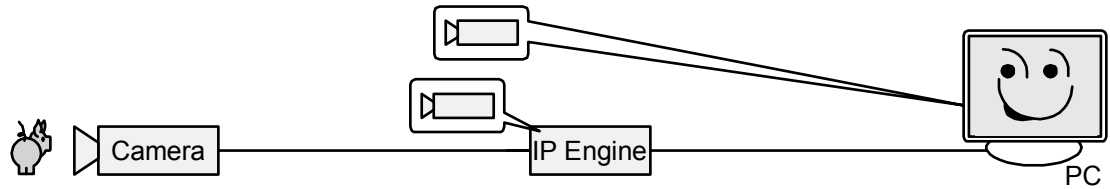
## Method 3: External sync, PC controlled

Camera: Triggered

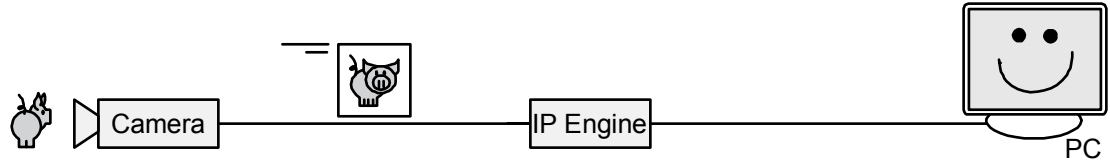
IP Engine: Triggered by PC

## 22 Grabbing (image acquisition)

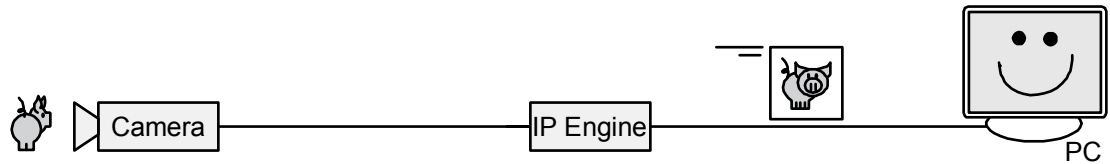
The PC requests an image from the IP Engine, which then requests an image from the camera.



The camera sends the image to the IP Engine...



...which then forwards it to the PC.

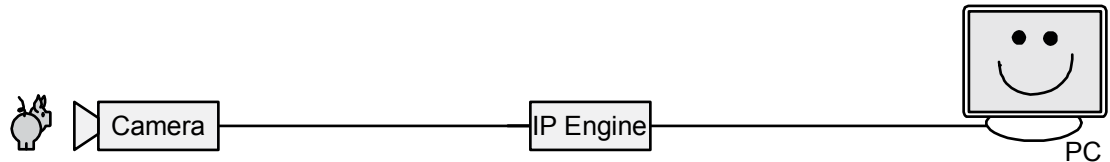


### Method 4: External sync, grabber controlled

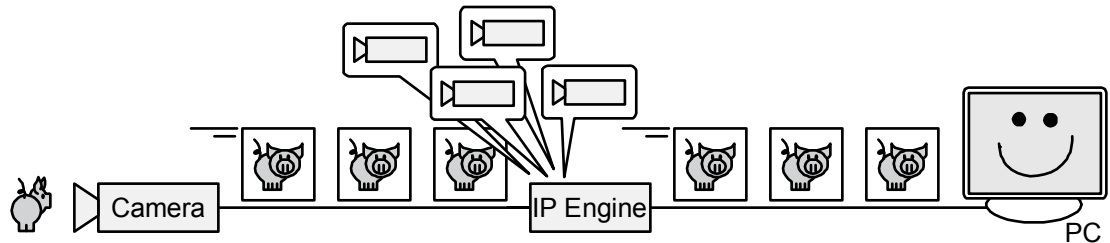
Camera: Triggered

IP Engine: Self-triggered

The IP Engine determines when to request an image from the camera. It may do so using the PLC (Programmable Logic Controller) alone, or in conjunction with an external trigger such as a presence detector.



Depending on how you configure the system, the IP Engine can request a single image, a burst, or a steady stream.



## In practice...

Your IP Engine supports many possible configurations for acquiring images. Thus, the links below provide useful information, but not exact directions. In general, the best place to start is the *iPORT Quick Start Guide*.

### To acquire images using method 1 (free run):

- See the *iPORT Quick Start Guide*.

**To acquire images using method 2 (free run, PC controlled) or method 3 (external sync, PC controlled):**

- See the *iPORT Coyote Software Guide (Configuration dialog)*.

**To acquire images using method 4 (external sync, grabber controlled):**

- See the *iPORT Programmable Logic Controller Reference Guide*.

**To acquire images programmatically (all methods):**

- See the *iPORT C++ SDK Reference Guide (CyGrabber)*.

## 24 Grabbing (image acquisition)

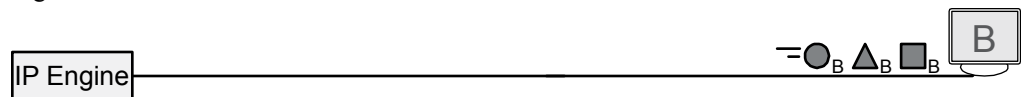
# Unicast, multicast, and multi-unicast modes

In addition to sending network packets to a single PC, your IP Engine can send the same packets to multiple PCs. The sending modes are called unicast, multicast, and multi-unicast.

The multiple modes are available for IP Engines with firmware versions 4.0 or higher.

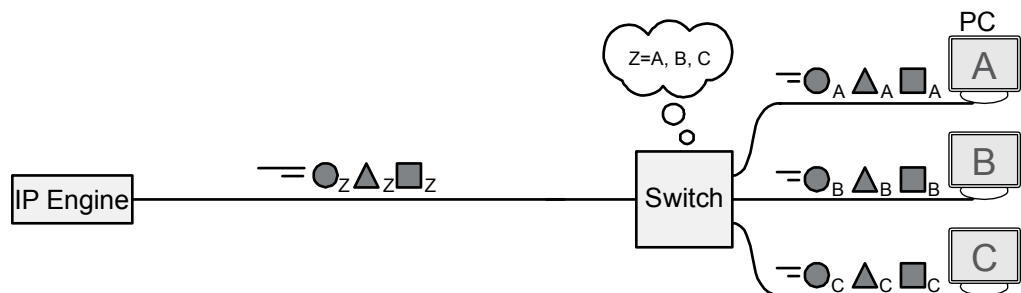
## Unicast

In unicast mode, the IP Engine sends images to a single PC, on request. This mode is the default data sending mode.



## Multicast

In multicast mode, the IP Engine sends each frame to a multicasting group, as triggered by the master PC.



Multicasting uses the IGMP (Internet Group Management Protocol) and requires an IGMP-compliant switch. The switch maintains a group list containing the group's IP address and the IP addresses of all the PCs that have subscribed to the group. The multicasting group's IP address is 224.64.xx.xx; where xx is configurable. When the switch receives a packet addressed to the group, it sends individual packets to all the (PCs) IP addresses in the list.

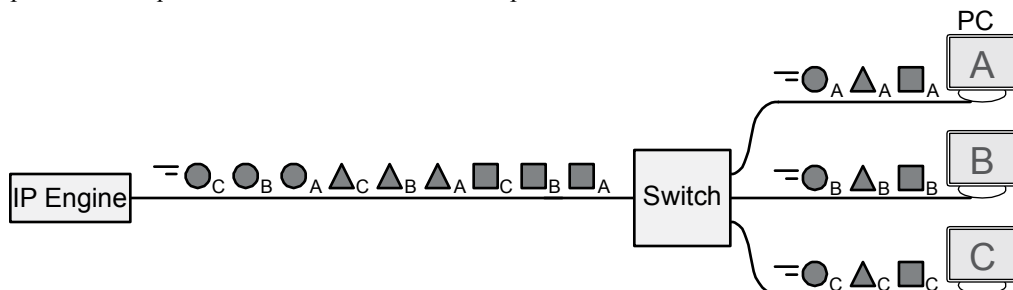
To learn more about IGMPv3, see *RFC 3376* at [www.ietf.org](http://www.ietf.org).

## Multi-Unicast

Multi-Unicast mode lets you send packets to multiple PCs without having an IGMP-compliant switch. Where the switch maintained a group list in multicasting mode, the IP Engine maintains the list for

## 26 Unicast, multicast, and multi-unicast modes

multi-unicasting. The IP Engine consecutively sends each packet individually to each of the PCs, so a group of  $n$  PCs requires  $n$  times the bandwidth compared to unicast.



Multi-unicast mode also lets you send a single image to a single PC, even if other PCs are grabbing continuously.

### In practice...

#### To configure your IP Engine in Coyote:

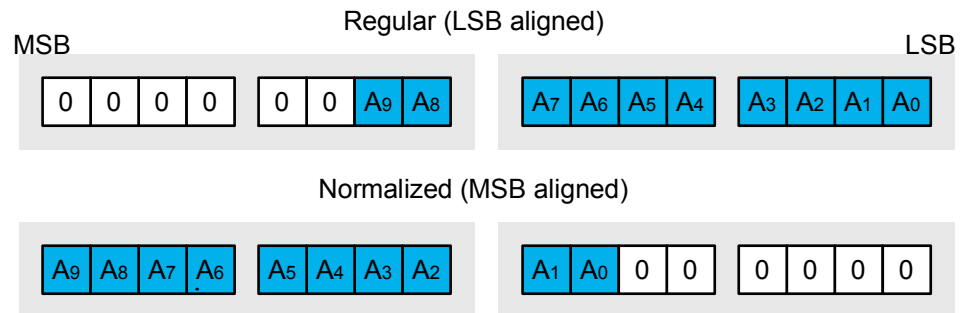
- See the *iPORT Coyote Software Guide* (**Multi-Target Configuration** dialog).

#### To configure your IP Engine programmatically:

- See the *iPORT C++ SDK Reference Guide* (`CyDeviceFinder::DeviceEntry::mSendingMode`).

# Pixel alignment

When your IP Engine sends a pixel to your PC, it can arrange the data in regular format or normalized format. If you're using Coyote, the choice isn't important. However, if you're processing the image using your own program, you need to configure the pixel type to match the format expected by your program.



## In practice...

### To configure pixel alignment in Coyote:

- See the *iPORT Coyote Software Guide* (**Configuration** dialog > **Pixel** tab).

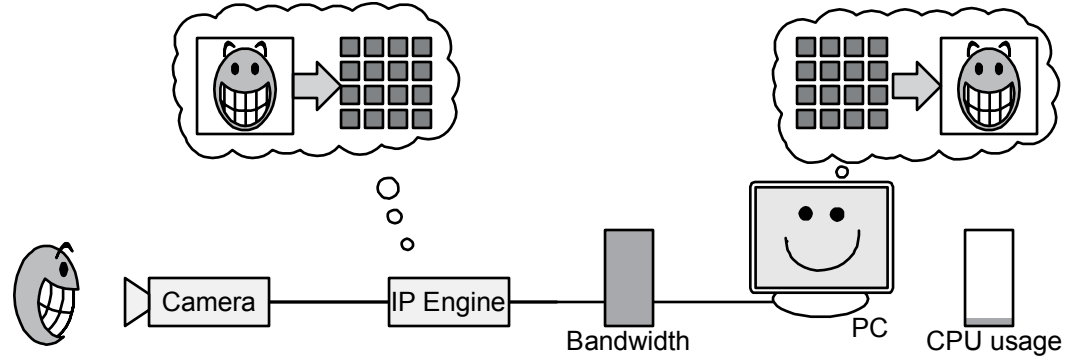
### To configure pixel alignment programmatically:

- See the *iPORT C++ SDK Reference Guide* (`CY_GRABBER_PARAM_NORMALIZED`).

## 28 Pixel alignment

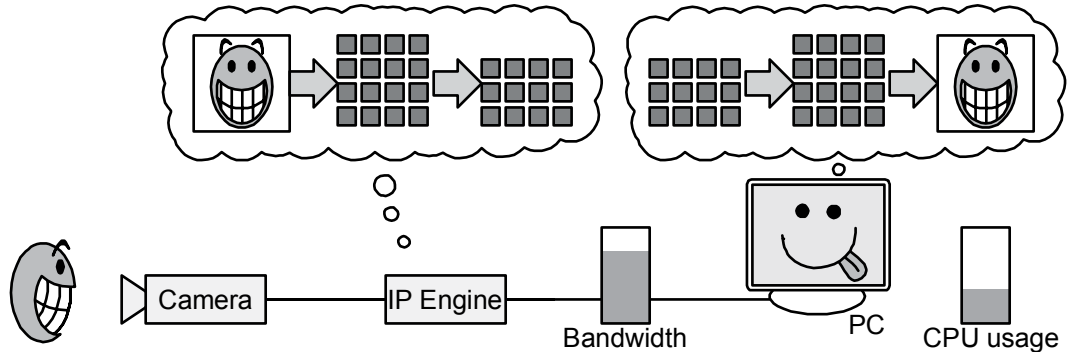
# Packing

When transmitting an image, the IP Engine breaks the image from the camera into packets and sends them to the host PC. The PC reassembles the packets into an image.



By default, the IP Engine saves each pixel in separate bytes. An 8-bit pixel requires one byte; a 10-bit or 12-bit pixel requires two bytes (16 bits); an RGB-24 pixel, four bytes (32 bits). The operation is quick, but, except for the 8-bit pixel, some bits are unused (wasted).

By packing the pixels more efficiently, the IP Engine can reduce the number of bytes by 25% without any loss of data. Thus, instead of sending four bytes, the IP Engine needs only transmit three, resulting in a 25% reduction in bandwidth between the IP Engine and the PC. However, most packing arrangements require more CPU to unpack (put the pixel back into an easily readable form).



## Bandwidth savings due to pixel packing

Pixel type	Unpacked size (bytes/pixel)	Packed size (bytes/pixel)	Bandwidth savings
8-bit	1	n/a	none
10-bit	2	1.5	25%
12-bit	2	1.5	25%
RGB-24	4	3	25%

## Packing arrangements

The actual arrangement of the packed pixels depends on the alignment of the pixel before packing. However, one pixel always uses the first 12 bits, the other, the second 12 bits. To learn about pixel alignment, see “Pixel alignment” on page 27.

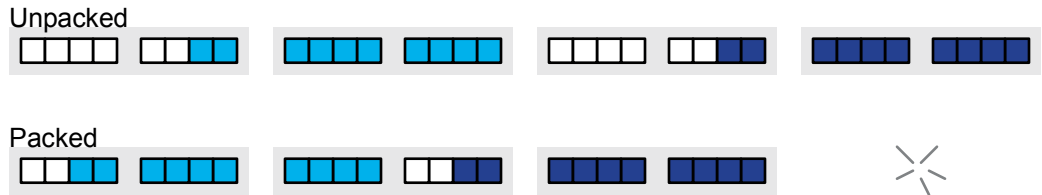
### 8-bit pixels

Since each 8-bit pixel uses exactly one byte, 8-bit pixels can't be packed more efficiently than they already are.

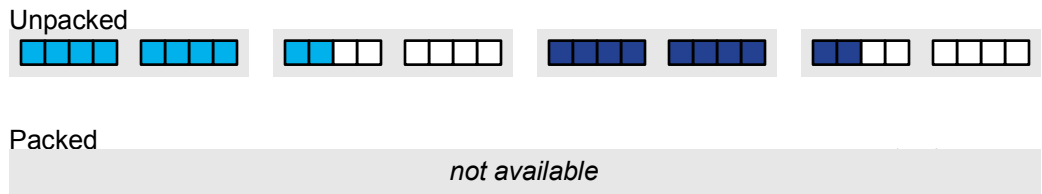
### 10-bit pixels

The IP Engine packs two 10-bit pixels into three bytes (24 bits). Though 4 bits are still unused, an arrangement that wasted no bits (e.g. four 10-bit pixels in 5 bytes) would be very computationally expensive to unpack.

#### 10-Bit Regular

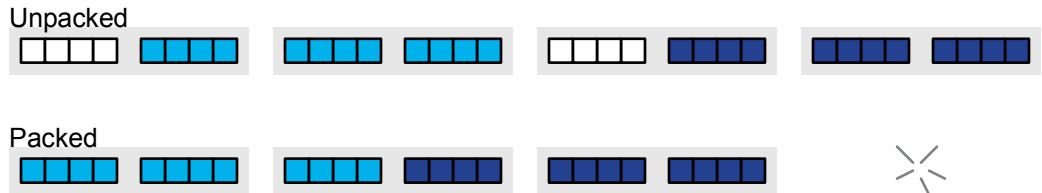


#### 10-Bit Normalized



### 12-bit pixels

#### 12-Bit Regular



## 12-Bit Normalized

Unpacked



Packed

*not available*

## RGB-24 pixels

RGB-24 pixels use one byte each for red, green, and blue. The last byte is unused, so packing incurs little or no CPU usage penalty.

### RBG-24

Unpacked



Red

Green

Blue

Packed



Red

Green

Blue

## In practice...

### To configure packing settings in Coyote:

- See the *iPORT Coyote Software Guide* (**Configuration** dialog > **Pixel** tab).

### To configure the packing settings programmatically:

- See the *iPORT C++ SDK Reference Guide* (`CY_GRABBER_PARAM_PACKED`).



# Binning, decimation, and area of interest

In this section:

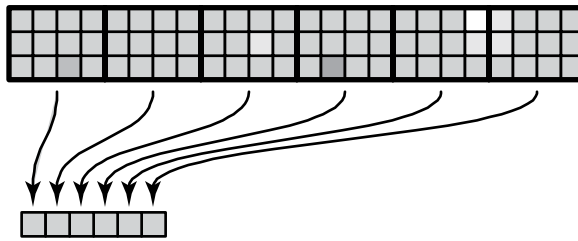
Understanding binning.....	33
Understanding decimation .....	33
Understanding area of interest .....	34
Order of binning, decimation and area of interest .....	34

## Understanding binning

Binning combines several pixels into a single pixel to improve the signal-to-noise ratio in the image (reduce the resolution, increase the dynamic range). If your camera supports binning, setting **Binning X** to 4 and **Binning Y** to 3 in Coyote combines 12 pixels into one “superpixel”.

Binning X = 4

Binning Y = 3



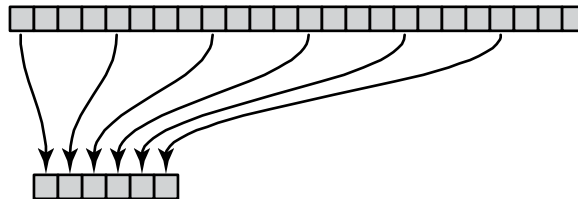
## Understanding decimation

Decimation typically involves discarding all pixels except for every  $n^{\text{th}}$  one. For example, in Coyote, a **Decimation X** value of 4 and a **Decimation Block X** value of 1 keeps 1 pixel and removes 3.

Decimation X = 4

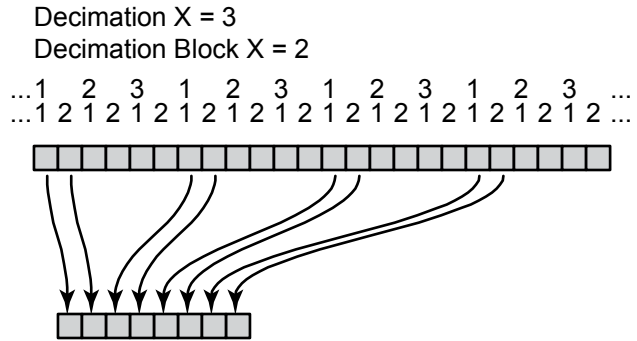
Decimation Block X = 1

... 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 ...



## 34 Binning, decimation, and area of interest

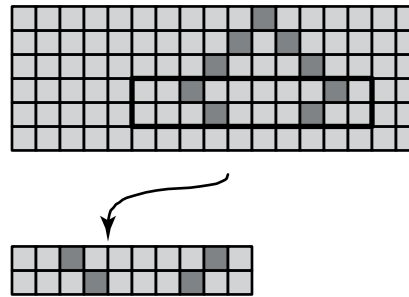
The IP Engine also lets you count by blocks of pixels as well as by single pixels. A **Decimation X** value of 3 and a **Decimation Block X** value of 2 keeps 2 pixels and removes 4.



## Understanding area of interest

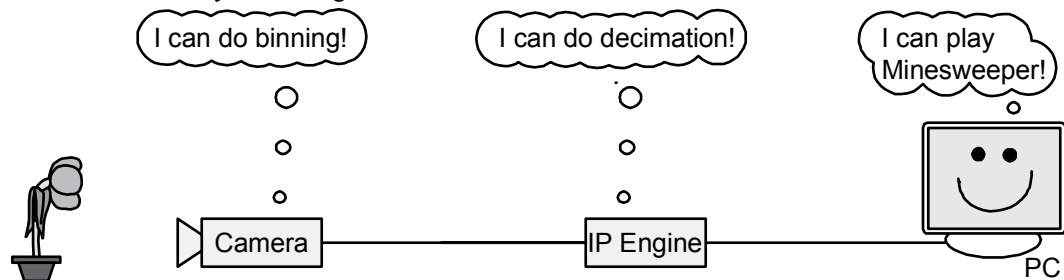
Area of interest lets you transmit only a portion of the image. In some cameras, reducing the area of interest lets you increase the frame rate.

Width = 10 Offset X = 5  
Height = 2 Offset Y = 3

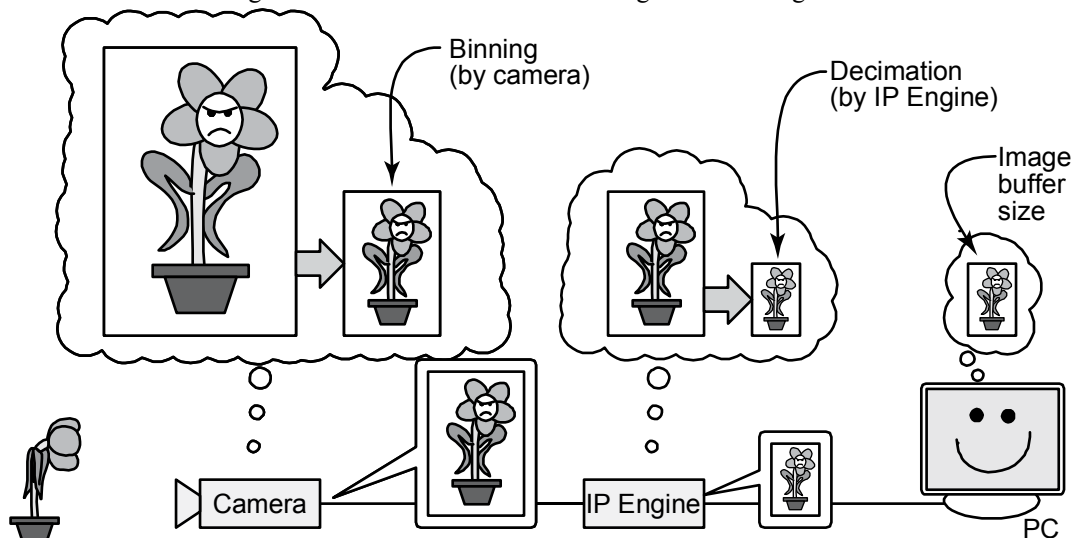


## Order of binning, decimation and area of interest

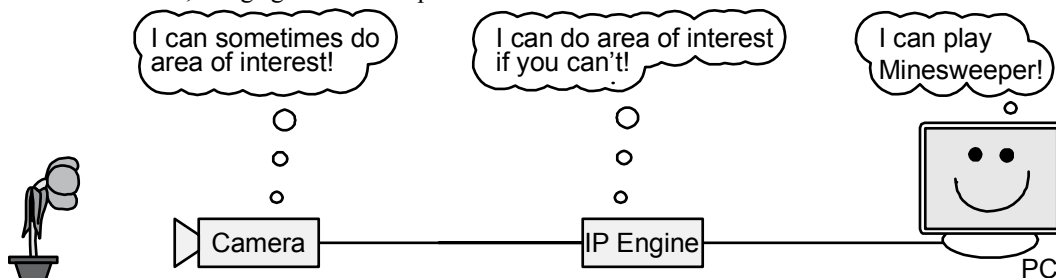
Binning, decimation, and area of interest all reduce the size of the image. However, the order in which they're applied affects the final outcome. Binning is done by the camera (if the camera supports it). Decimation is done by the IP Engine.



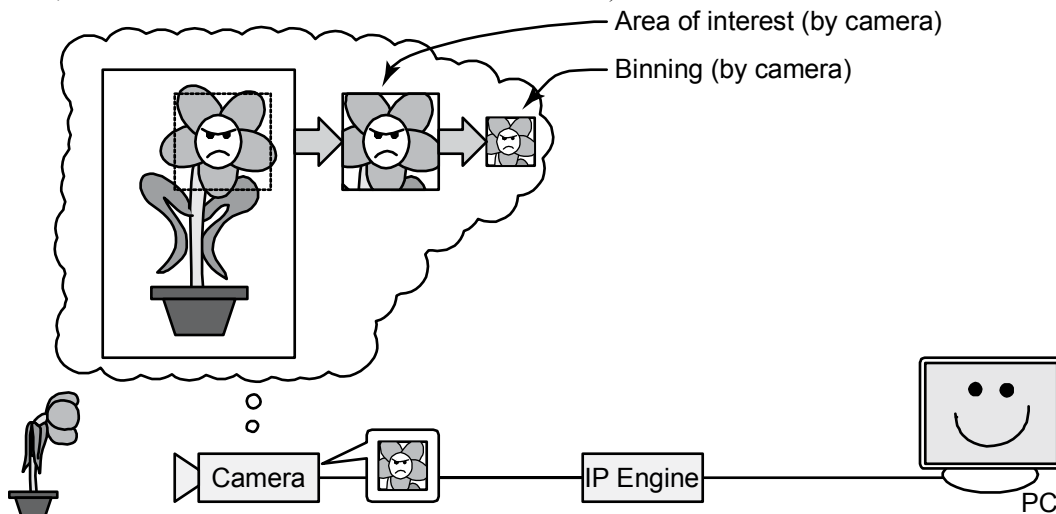
The size of the final image buffer in the PC is that of the image after binning and decimation.



For area of interest, things get more complicated.

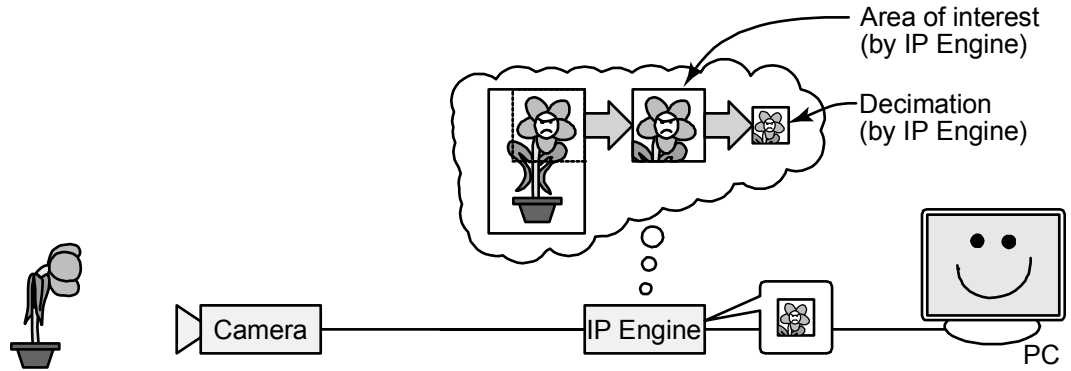


Cameras that support area of interest *typically* calculate area of interest first, then binning. (To be certain, consult the camera manufacturer's documentation.)



## 36 Binning, decimation, and area of interest

If the camera doesn't support area of interest, the IP Engine calculates the area of interest, followed by decimation.



### In practice...

#### To configure binning, decimation, and area of interest in Coyote:

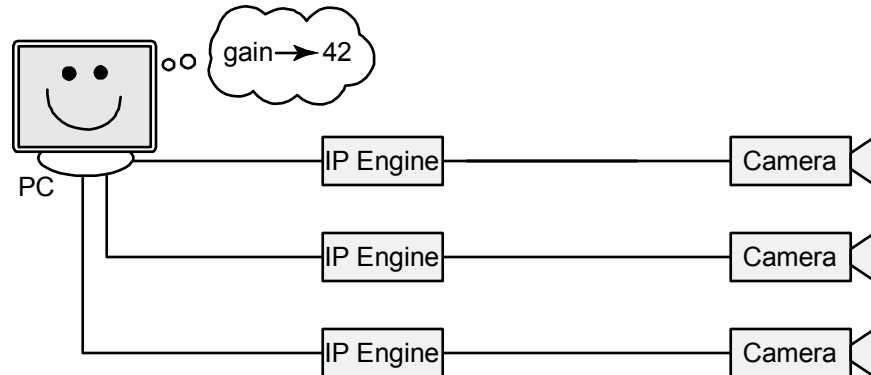
- See the *iPORT Coyote Software Guide* (**Configuration** dialog > **Image** tab).

#### To configure settings programmatically:

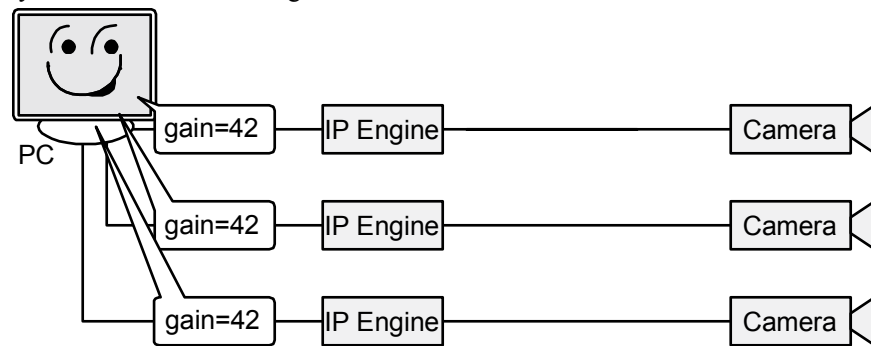
- See the *iPORT C++ SDK Reference Guide* (the `CY_GRABBER_PARAM_DECIMATION`, `CY_GRABBER_PARAM_BINNING`, and `CY_GRABBER_PARAM_OFFSET` family of parameters).

# GigE Vision

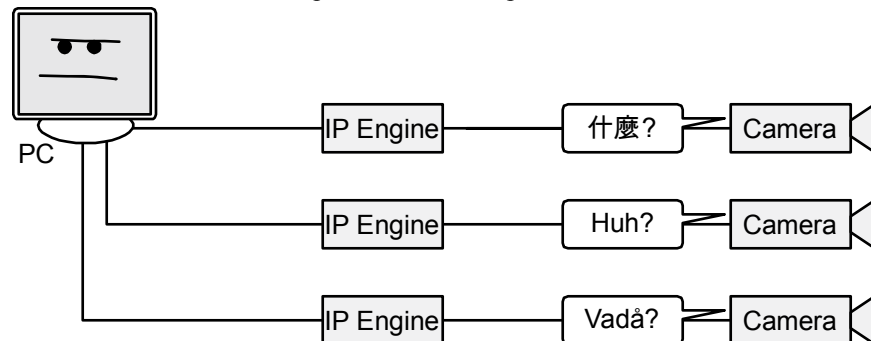
Ideally, knowing how to set a parameter on one camera...



...would let you make the same setting for all cameras.

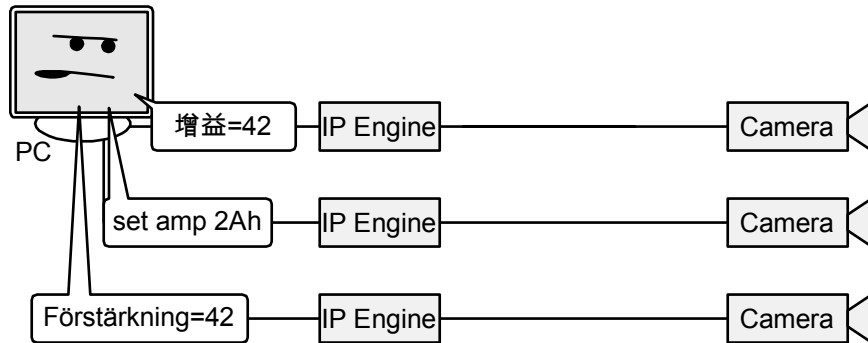


However, cameras have manufacturer-specific or model-specific instruction sets.

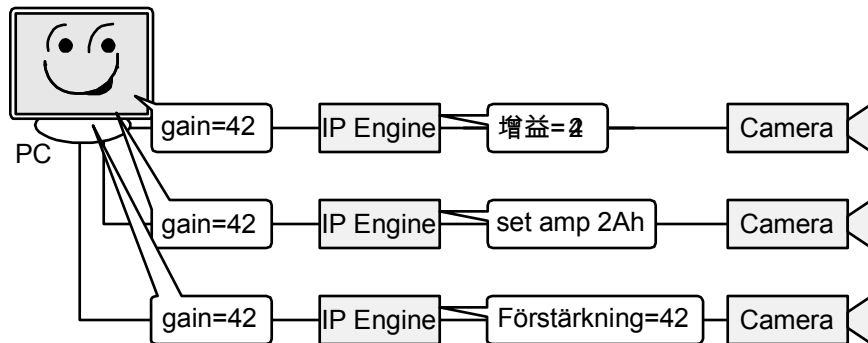


## 38 GigE Vision

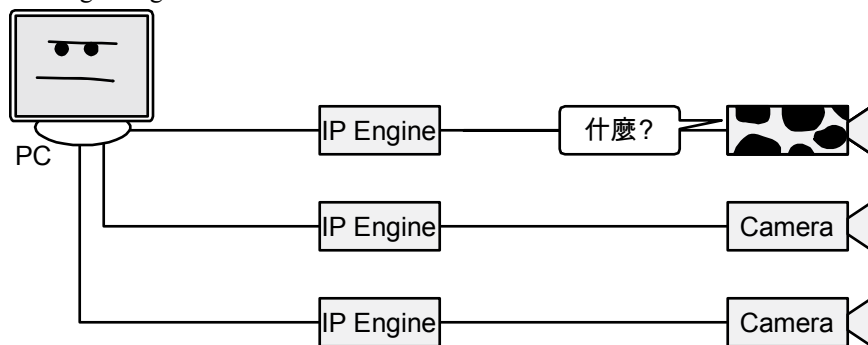
As a result, end users, programmers, or *someone* had to adapt for every camera they used.



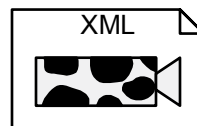
To reduce frustration, Pleora created camera-specific DLLs for as many models as they could. These DLLs helped provide a more standard interface. Though the process was time-consuming (for Pleora), the DLLs worked well...



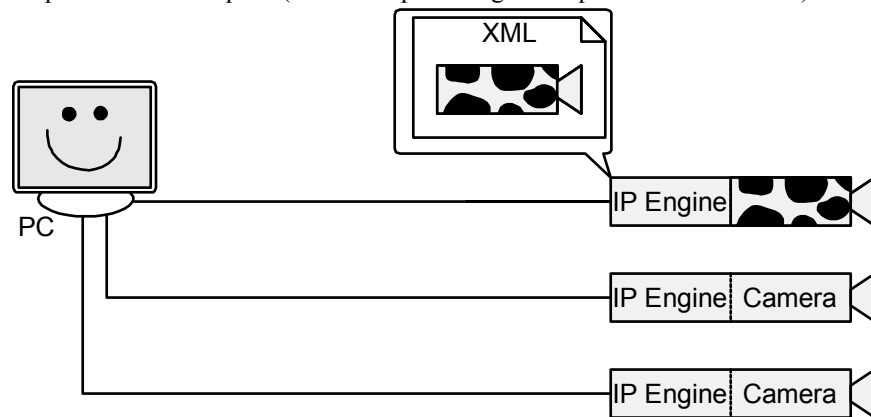
...unless something changed.



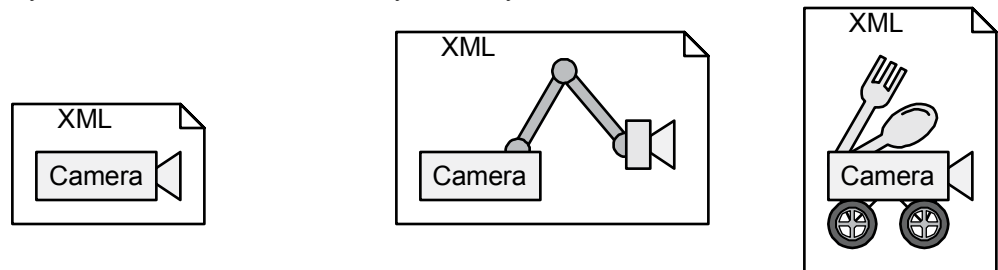
Recognizing that the increasing complexity of modern cameras would exacerbate end-user frustration (and that keeping all DLLs up-to-date was impossible), Pleora and a group of companies created the GigE Vision Standard. To be compliant, a GigE Vision device has to provide an XML file that defines its features and how to use them.



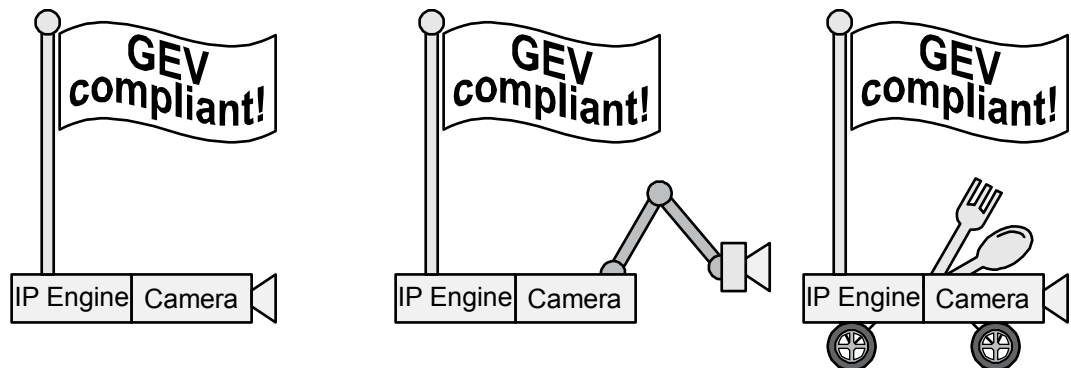
Using an iPORT IP Engine and a camera together, you can create a system that behaves as a single, GEV-compliant device *without having to make any changes to the camera*. The IP Engine stores the XML file and provides it on request (as well as providing the required GEV interface).



If you're a camera manufacturer, you can use iPORT AutoGEV to create the GEV interface. As well as configuring the GEV interface, AutoGEV lets you automatically create XML files that define all the features on your camera... no matter how many features your camera has.



Using iPORT AutoGEV and your iPORT IP Engine, you can quickly make your existing cameras GEV-compliant so you can get to market quickly.



(Contact your Pleora sales representative for details and pricing for iPORT AutoGEV software.)

## In practice...

### To use a GEV-enabled IP Engine:

1. Install the iPORT PureGEV Suite.
2. Follow the directions in the *iPORT PureGEV Quick Start Guide*.



# Index

## A

area of interest 34  
AutoGEV 39

## B

bandwidth  
    minimizing 15, 25, 29  
    throttling 19  
binning 33  
BOOTP 12

## C

camera  
    free run 21  
    image sizing 33  
    triggered 21  
CPU usage  
    minimizing 15, 19

## D

decimation 33  
DHCP server 9

## F

flow control 19

## G

GigE Vision 37  
grabbing 21

## H

heartbeat 17

## I

IGMP 25  
image  
    acquisition 21  
    reducing size 33  
interpacket delay 19  
IP address

    multicast group 25  
    static 12  
    unique 9

## IP Engine

    assigning an IP address 12  
    detecting network failure 17  
    detection 11  
    GigE Vision enabled 37  
    image acquisition 21  
    image sizing 33  
    image transmission 25  
    packing 29  
    pixel format 27

## IPv4 15

## J

jumbo packet 15

## L

layer 2 9  
layer 3 10

## M

MAC address 9  
MTU 15  
multicast 25  
    heartbeat 17  
multi-unicast 25

## N

network failure  
    recovering 17  
network flow control 19

## P

packet  
    configuring size 16  
    interpacket delay 19  
    jumbo 15  
    preventing loss 13

- packing 29
- pixel
  - alignment 27
  - packing 30
  - types 29
- PLC 22
- power outage
  - recovering 17

## **R**

- RFC 1122 13
- RFC 3376 25

## **S**

- subnet masks 13
- subnets 13
- superpixel 33
- switch 9
  - IGMP-compliant 25
  - supported packet size 15

## **T**

- triggering 21

## **U**

- unicast 25
- unpacking 29