

Programming EPICS with PERL

Mohan Ramanathan

October 12, 2004

Part of the EPICS “Getting Started” Lecture Series

Argonne National Laboratory



*A U.S. Department of Energy
Office of Science Laboratory
Operated by The University of Chicago*



Outline

- **History**
- **Quick Overview of Perl**
- **EPICS Overview**
- **Channel Access (EZCA)**
- **Perl interface to EZCA**
- **Perl Applications**
- **Current and Future**
- **Conclusions**
- **Acknowledgments**



History

- **Perl is an easy language for WEB programming (CGI)**
 - Needed an easy to program , yet powerful language to write code to interface to CA, SDDS and for writing CGI interface
- **Object oriented**
- **Easy to interface to other languages like C, C++, java etc..**
- **Preexisting objects modules are available in abundance**
<http://www.cpan.org>
- **Perl is free!**
- **Great support and free!**



History

- **Why Perl?**
 - Back in 1995/96 was looking for a language to use for various applications
 - C was too cumbersome
 - Needed a quick and easy to use, yet a powerful language
 - Tcl/Tk was used at APS but was not properly suited for the applications I was interested in (mainly WEB interface)
 - Easy to learn !!
 - Perl has simple constructs and looks similar to C
 - Numerous modules available in CPAN for graphics and GUI
 - pTK was already available for Perl in 1996
 - Interpreter based language means no compilation
 - Full programs can be generated easily with Perl
- **Perl EZCA module was born in 1996**



Quick Overview of Perl

- **A high-level interpreted programming language**
- **Modern programming language**
 - Data Structures
 - Control Structures
 - Regular Expressions
 - Object Oriented
- **Perl handles both strings and numbers elegantly**
 - Excellent for manipulation of both strings and numbers
 - Most string and arithmetic operators are similar to C
 - All operators and most functions work on scalar or array data of any type
- **Multi-platform support**
 - Unix, Linux, Windows, etc..

Quick Overview of Perl

- **Data types**
 - Scalars need no assignment. Internally all scalars are assigned to double. However for users it takes the form of the data it holds
 - Arrays can be multi-dimensional. However internally it is only one dimension
- **Variable Names**
 - All scalars start with \$
 - All arrays start with @
 - All hash reference start with %
- **Has special variables and arrays like \$_, @_, @ARGV, %ENV, etc..**
- **User has a choice of (not) using special cryptic variables !!**
- **Memory allocation is dynamic – will use up all memory when needed !**

Quick Overview of Perl - Operators

- **Binary and String Operators**

```
$a = 123; $b = 456;  
$c = 3;  
print $a + $b;      # prints 579          (addition)  
print $a . $b;      # prints 123456       (concatenation)  
print $a x $c;      # prints 123123123   (repeat)
```

- **Logical Operators**

```
$a && $b             # $a if $a is false, $b otherwise  
! $a                 # True if $a is not true
```

- **Numeric and String Comparison Operators**

```
$a == $b            # True if $a is equal to $b   (numeric)  
$a eq $b            # True if $a is equal to $b   (string)
```

- **File Test Operators**

```
-e $a                # True if file named in $a exists
```

Quick Overview of Perl - Control

- **if and unless statements**

```
if ($level < 0) {  
    print "Level is below Zero \n";  
}  
print "City is not New York \n" unless ($city eq "New York");
```

- **while and until statements**

```
while ($level < 100) {  
    $level++;  
    print "level value is $level\n";  
}  
until ($level == 0) {  
    $level--;  
    print "level value is $level\n";  
}
```

Quick Overview of Perl - Control

- **for statements**

```
for ($level =0; $level <100; $level++) {  
    print "level value is $level\n";  
}
```

- **foreach statements**

```
foreach $user (@users) {  
    print "$user is ok\n";  
}
```

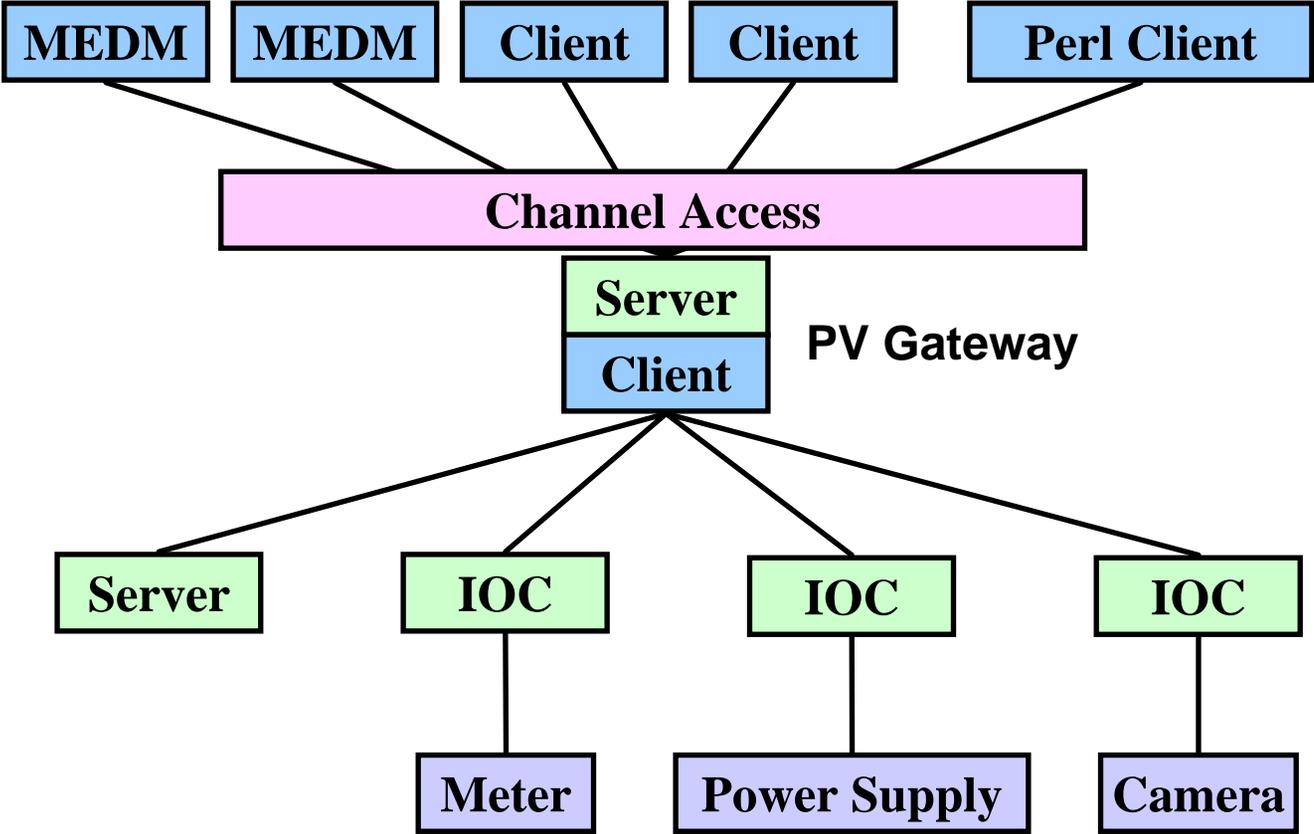
- **next**

```
foreach $user (@users) {  
    next unless ($user eq "jack");  
}
```

- **last**

```
foreach $user (@users) {  
    last if ($user eq "jack");  
}
```

EPICS Overview



Channel Access (EZCA)

EPICS extension for “E-Z (Easy) Channel Access”

- **EZCA provides a simplified interface to Channel Access, which is useful for some C programs which do not need all the capabilities of the full API**
- **Don’t need to handle chids, just use PV name strings – hash table**
- **Synchronous APIs - applications don’t have to handle callbacks**
- **Has only 25 routines to handle synchronous groups, individual requests and easy error handling**

Perl Interface to EZCA

Description

- **EZCA.pm is a Perl interface to EPICS EZCA library**
- **The routine names are based on EZCA nomenclature**
- **Each routine name in EZCA has an equivalent in Perl module**
- **All the commands are similar to what is used in EZCA**
- **Data types as described in EZCA (ezctype) are used**
 - *ezcaByte*
 - *ezcaString*
 - *ezcaShort*
 - *ezcaLong*
 - *ezcaFloat*
 - *ezcaDouble*

Perl Interface to EZCA

Return Codes

- Most function calls return status codes indicating the success/failure of the call
- Following are the return codes
 - *EZCA_OK*
 - *EZCA_INVALIDARG*
 - *EZCA_FAILEDMALLOC*
 - *EZCA_CAFailure*
 - *EZCA_UDFREG*
 - *EZCA_NOTCONNECTED*
 - *EZCA_NOTTIMELYRESPONSE*
 - *EZCA_INGROUP*
 - *EZCA_NOTINGROUP*

Perl Interface to EZCA

Work in Progress

- **Most functions with the exception of the Group function have been implemented and tested**
 - **21 of the 25 functions are working**
- **All functions involving arrays have a limitation of 2000 elements**
- **In the case of PV Put at this time only one element is possible**

Perl Interface to EZCA

Main Functions

- `@data = EZCA::Get($pvname, "ezctype", $no_elem)`

Retrieves values from *\$pvname* and the data type is specified as *ezctype*

The return status code is stored in *\$data[0]* and the retrieved values are in the rest of the array starting at elements *\$data[1]*

At this time the no of elements *\$no_elem* is limited to 2000

- `$status = EZCA::Put($pvname, "ezctype", $no_elem, $data)`

Write values into *\$pvname*.

The value is send in *\$data* and the data type is specified as *ezctype*.

The return status code is *\$status*

At this time the no of elements *\$no_elem* is limited to 1

- `EZCA::GetStatus($pvname, $tstamp, $tnstamp, $status, $severity)`

Retrieves various information about the said *\$pvname*.

The time in seconds is in *\$tstamp* and in nanoseconds is in *\$tnstamp*.

The *\$status* and *\$severity* returns status and the severity of the process variable

Perl Interface to EZCA

Associated Functions

- `$status = EZCA::GetNelem($pvname, $nelem)`
The number of elements in the *\$pvname* is returned in *\$nelem*.
- `$status = EZCA::GetPrecision($pvname, $prec)`
The precision for the *\$pvname* is returned in *\$prec*.
- `$status = EZCA::GetUnits($pvname, $unit)`
This returns the Engineering Unit field of the *\$pvname* in *\$unit*.
- `$status = EZCA::GetControlLimits($pvname, $low, $high)`
Retrieve the Control Limits values in *\$low* and *\$high*.
- `$status = EZCA::GetGraphicLimits($pvname, $low, $high)`
Retrieve the Graphics Limits values in *\$low* and *\$high*.

Perl Interface to EZCA

Monitors

- `$status = EZCA::SetMonitor($pvname, "ezctype")`
Used to set a monitor on the *\$pvname*.
- `$status = EZCA::ClearMonitor($pvname, "ezctype")`
Used to clear any monitors placed on the *\$pvname*.
- `$mon = EZCA::NewMonitorValue($pvname, "ezctype")`
Use this function to poll for new values on a previously set monitor.
A non zero return value in *\$mon* indicates a new value since the last time the value was read via a Get
- `$status = EZCA::Delay($seconds)`
Use to introduce delays between calls.
The time *\$seconds* is in seconds. Fractional time are allowed.

Perl Interface to EZCA

Error Handling

- `EZCA::AutoErrorMessageOn()`
Turn ON the EZCA error messages to be printed on the *stdout*.
- `EZCA::AutoErrorMessageOff()`
Turn OFF the EZCA error messages to be printed on the *stdout*.
- `EZCA::Perror($prefix)`
Use this to optionally add a prefix message to the EZCA error message.
- `EZCA::GetErrorString(NULL, $stringBuff)`
Retrieve the EZCA error message in the *\$stringBuff* instead of *stdout*.
- `EZCA::Free($stringBuff)`
Use this to free up the *\$stringBuff* used in the above call.

Perl Interface to EZCA

Tuning EZCA

- `$seconds = EZCA::GetTimeout()`
Retrieve the current CA timeout value in seconds
- `$number = EZCA::GetRetryCount()`
Retrieve the current CA number of retries.
- `$status = EZCA::SetTimeout($seconds)`
Set the current CA timeout value in seconds. *\$status* is the return status
- `$status = EZCA::SetRetryCount($number)`
Set the CA number of retries. *\$status* is the return status

Perl Applications

Use EZCA and GD modules to generate images every minute for WEB

```
#!/usr/bin/perl
#####
#       Program to retrieve the data from EPICS and generate
#       png files of Storage ring history.
#####
use Time::CTime;          # load Time module for manipulating time
use EZCA;                 # load EZCA module
use GD;                   # load GD graphics module
use File::Copy;           # load File handling modules

$ENV{'EPICS_CA_AUTO_ADDR_LIST'} = 'NO';
$ENV{'EPICS_CA_ADDR_LIST'} = "164.54.188.65"; #environment for Rhea gateway

$epoch = 631152000;       # this is the difference between EPOCHS

$font   = "/usr/openwin/lib/X11/fonts/TrueType/Arial.ttf";
$bfont  = "/usr/openwin/lib/X11/fonts/TrueType/Arial-Bold.ttf";
$bifont = "/usr/openwin/lib/X11/fonts/TrueType/Arial-BoldItalic.ttf";

$smallfile = "/Public/aod/blops/plots/smallStatusPlot.png";
$tempfile  = "/Public/aod/blops/plots/statusPlot.png";
$pdajpgfile = "/Public/aod/blops/plots/tinyStatusPlot.jpg";
```

For PV
Gateway

Perl Applications

Use EZCA and GD modules to generate images every minute for WEB

```
EZCA::AutoErrorMessageOff();      # Turn off error messages to STDOUT
EZCA::SetTimeout(0.01);           # Fine Tune EZCA
EZCA::SetRetryCount(10);          # Fine tune EZCA
# setup monitor to trigger whenever this changes....
$val = EZCA::SetMonitor("S:SRdateCP","ezcaLong");
$val = EZCA::NewMonitorValue("S:SRdateCP","ezcaLong");
@pvvalues = EZCA::Get("S:SRdateCP","ezcaLong",1440);
EZCA::Delay(2.0);                 # Wait 2.0 seconds
# wait for the monitor to trigger action. happens every minute.
while(1) {
    $status = EZCA::NewMonitorValue("S:SRdateCP","ezcaLong");
    if ($status) {
        &getPV;                    # to retrieve the PVs
        &generateImage;            # generate the Images
        &generateHtml;            # generate the Html file
    }
    EZCA::Delay(5.0);             # Wait 5.0 seconds before polling
}
exit;
```

Important!
Performance
Issue

Need to poll
this only

Perl Applications

Use EZCA and GD modules to generate images every minute for WEB

```
sub getPV {  
# get the current status of storage ring ...  
    @pvvalues = EZCA::Get("S:IOC:timeOfDayForm1SI","ezcaString",1);  
    $lastUpdate = $pvvalues[1];  
    @pvvalues = EZCA::Get("XFD:srCurrent","ezcaFloat",1);  
    $current = sprintf("%6.1f mA", $pvvalues[1]);  
    @pvvalues = EZCA::Get("XFD:FillNumber","ezcaShort",1);  
    $fill_no= sprintf ("%4d", $pvvalues[1]);  
# get the time and current 24 hr data arrays ...  
    @timearray = EZCA::Get("S:SRdateCP","ezcaShort",1440);  
    shift @timearray;  
    @userarray = EZCA::Get("S:UserOpsCurrent","ezcaFloat",1440);  
    shift @userarray;  
# Find the time from the last data point and adjust for epoch ...  
    $curtime = $timearray[$#timearray]+$epoch;  
    $curhour = strftime("%H",localtime($curtime));  
    $curmin = strftime("%M",localtime($curtime));
```

**Difference
between EPICS
& UNIX EPOCHS**

Perl Applications

Use EZCA and GD modules to generate images every minute for WEB

```
sub generateImage {
# Initialize the size of the image...
    $timage = new GD::Image (360, 340);

# Allocate colors ...
    $twhite = $timage->colorAllocate (255,255,255);
    $tgreen = $timage->colorAllocate (0,170,0);
    $tblue = $timage->colorAllocate (0,0,255);

# Define the origin of the plot (0,0) of the image is the top left corner
    @torigin = (40,290);

# Generate the heading for the plot ..
    $timage ->stringTTF ($tblack,$bfont,10,0, 190,49,"Active Beamlines :");

# Generate the heading for the plot ..
    open (IMGFILE, ">$tempfile");
    print IMGFILE $timage->png;
    close IMGFILE;
```

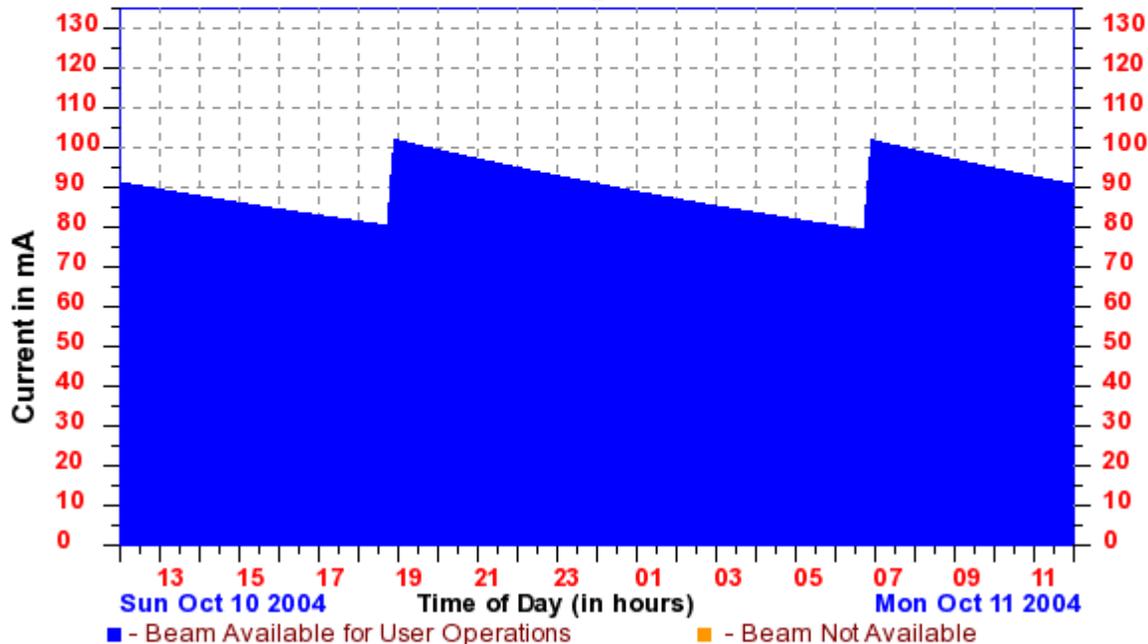
Perl Applications

Use EZCA and GD modules to generate images every minute for WEB

APS Storage Ring Status 13:01:00

Storage Ring Current : 90.9 mA **Beam Lifetime** : 46.2 hrs
Operating Mode : *Delivered Beam* **Global Feedback** : ON **Local Steering** : OFF
Message from Operations: **Beamlines Operating** : 35

Operator in Charge : Hill, Forth and Gagliano
Floor Coordinator : Lisa Gades (2-0101)
Fill Pattern : 0+324x1/1% Coupling
Problem Info :
Dump/Trip Reason :
Next Fill Info : Machine Studies @16:00



Perl Applications

Example of writing to an EPICS process variable ...

```
#!/usr/bin/perl
#####
#       Program to scan the Undulator from Closed gap to Open gap
#       and collect the encoder values at fixed gaps.
#####

use Time::CTime;          # Module needed to format time is fancy formats
use EZCA;                # Loads the EZCA.pm module
use English;             # this is to use $UID instead of $<
use Sys::Hostname;      # to get access to system variables

.
.

while ( abs($gap-$cgap) > 0.020 ) {
    $status = 1;
    until ($status == 0) {
        $status = EZCA::Put("ID$sector:ScanGap","ezcaFloat",1,$gap);
        EZCA::Delay(0.05);
    }
}
```

Perl Applications

Example of a program using SDDS and GD module

```
#!/usr/bin/perl
#####
#   Program to read the data files and generate a large gif file
#   of Storage ring history for the past 7 days.
#####

$dir = "/home/helios/XFDOPS/monitoring/shutter";
$tempfile = "/home/helios/xfdsys/monitoring/tempfiles/weektemp.sdds";
$pngfile2 = "/net/epics/Public/aod/blops/plots/WeekHistory.png";

use Time::CTime;
use SDDS;
use File::Copy;
use GD;

#   get the current time and the time 7 days ago ....
#   then get it nearest 0, 15, 30 or 45 minute interval.

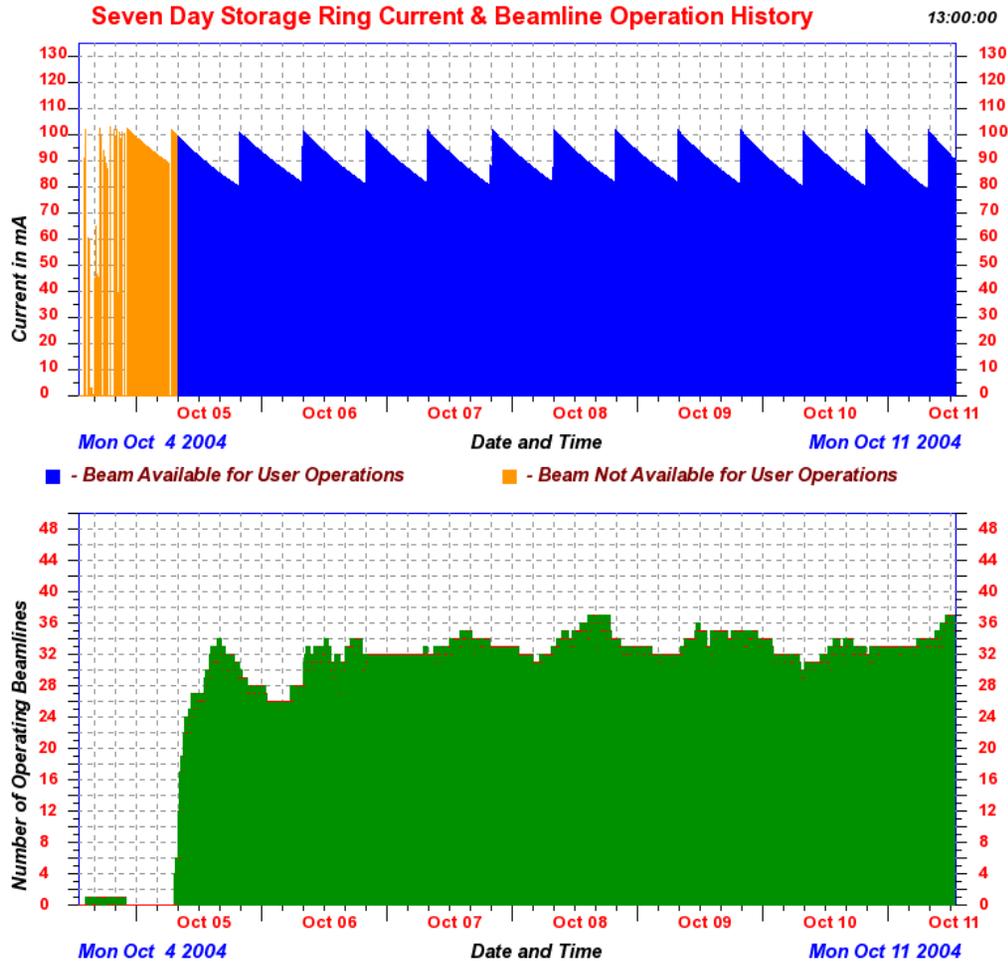
$curtime = time;

$curmin = strftime("%M",localtime($curtime));
$cursec = strftime("%S",localtime($curtime));
```



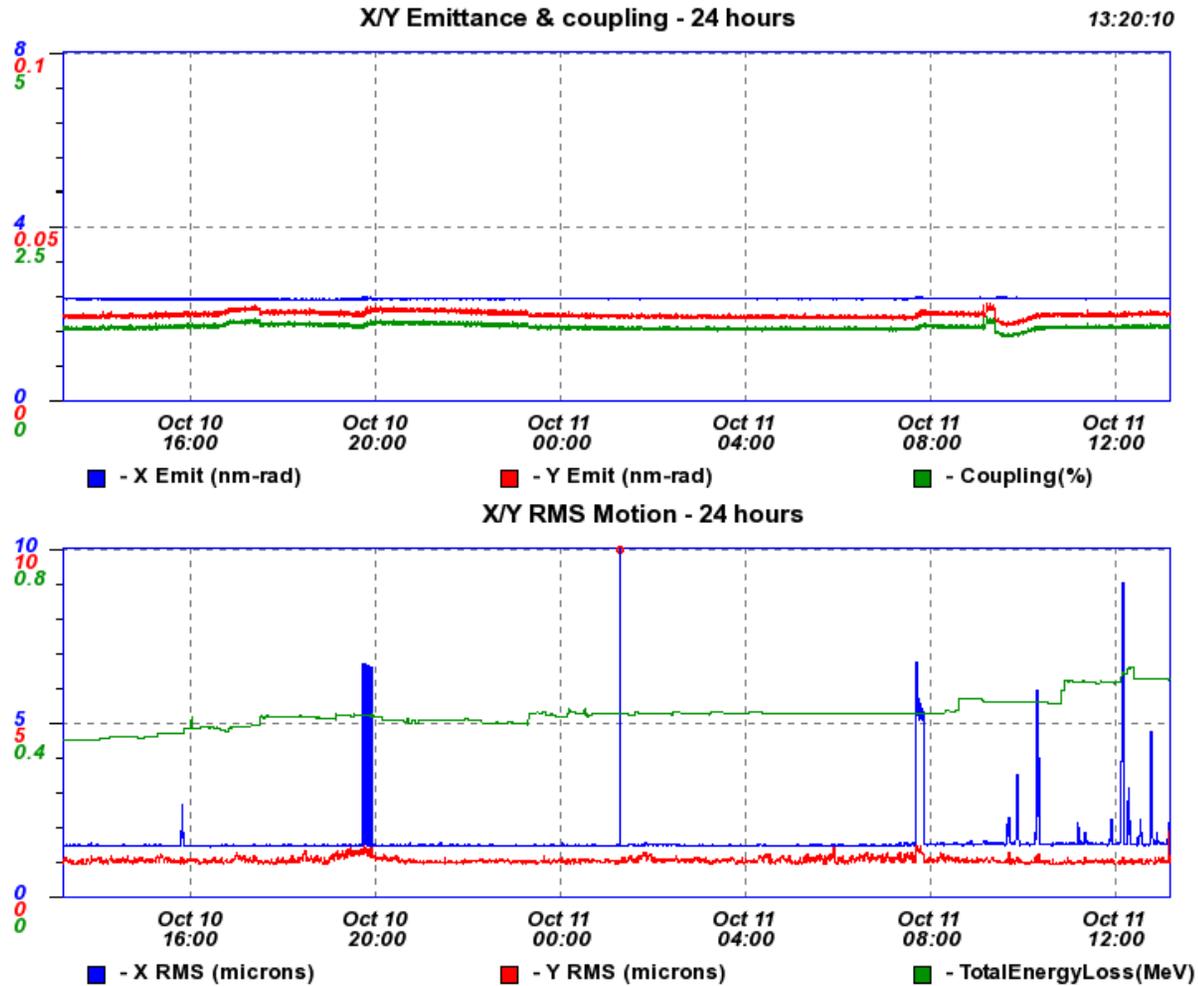
Perl Applications

Use SDDS and GD modules to generate this image for WEB



Perl Applications

Use SDDS and GD modules to generate this image for WEB



Current and Future

- **In use for over 8 years by numerous people in APS and others. Easy to develop quick applications.**
- **With the CA and other available Perl modules it is possible to create complex applications for the EPICS environment.**
- **Needs additional work in Linux port (bug fixes)**
- **Needs to be ported and tested with EPICS R3.14**
- **Built and test in Windows environment**
- **Interface directly to portable CA instead of EZCA**

Conclusions

How to get the Software

- Documentation available
<http://www.aps.anl.gov/aod/people/mohan/EZCA/EZCA.html>
- Installation packages for Solaris and Linux
- Source code available
- Requires EPICS libraries
- The software is available as gzipped tarfiles
<http://www.aps.anl.gov/aod/people/mohan/EZCA/soIEZCA.tar.gz>
<http://www.aps.anl.gov/aod/people/mohan/EZCA/linuxEZCA.tar.gz>
- For building it standalone, some of the EPICS include files and libraries are provided. These were build on Sun Solaris and Linux platforms

Acknowledgements

- **Karl Glazebrook the developer of PGPERL, the perl interface to PGPLOT for the help in dealing with arrays.**
- **Marty Kraimer for his help during the development days.**

What is PERL?

Practical Extraction and Report Language

OR

Pathologically Eclectic Rubbish Lister