MATLAB Programmer’s Guide for FACET physicists

(Version 0.81)

LCLS version: 9/22/08

small changes for FACET April, 2011  jrock
# REQUIREMENTS

## BASICS

1.1 Setting up the environment ......................................................... 5
1.2 Starting MATLAB ........................................................................ 5
1.3 Directories ................................................................................... 5
1.4 CVS ............................................................................................... 5
1.4.1 Basic concept ........................................................................... 6
1.4.2 CVS Commands ........................................................................ 6
  1.4.2.1 Checkout ............................................................................. 6
  1.4.2.2 Commit ............................................................................... 6
  1.4.2.3 Update ............................................................................... 6
  1.4.2.4 Add ..................................................................................... 6
  1.4.2.5 Remove ............................................................................... 6
  1.4.2.6 Diff .................................................................................... 7
  1.4.2.7 Other commands ................................................................ 7
1.4.3 CVSWEB ................................................................................... 7
1.4.4 Guidelines ............................................................................... 7

## LABCA

2.1 General concepts ......................................................................... 8
  2.1.1 Parameters and return values ............................................... 8
  2.1.2 Type ..................................................................................... 8
  2.1.3 Timestamp format ............................................................... 8
  2.1.4 Exceptions ........................................................................... 8
  2.1.5 Timeouts ............................................................................. 9
2.2 Basic commands .......................................................................... 9
  2.2.1 lcaGet .................................................................................. 9
  2.2.2 lcaPut, lcaPutNoWait .......................................................... 10
2.3 Monitors ..................................................................................... 11
  2.3.1 lcaSetMonitor ..................................................................... 12
  2.3.2 lcaNewMonitorValue ......................................................... 12
  2.3.3 lcaClear .............................................................................. 13
2.4 Network-related settings ............................................................ 13
  2.4.1 Timeout ............................................................................... 14
  2.4.2 Retry count .......................................................................... 14
  2.4.3 Default configuration ......................................................... 15
2.5 Other ......................................................................................... 15

## AIDA

3.1 Java setup .................................................................................. 16
3.2 Browsing names ......................................................................... 16
3.3 Retrieving data .......................................................................... 16
3.4 Miscellaneous ............................................................................ 18

## Beam Synchronous Acquisition

4.1 Reserving an event definition .................................................... 19
4.2 Changing default parameters ................................................... 20
4.3 Starting data acquisition ......................................................... 20
4.4 Releasing an event definition .................................................. 21
4.5 An example script .................................................................... 21

## Image Management

6.1 Overview ................................................................................... 23
6.2 Global data ................................................................................ 23
6.3 GUI Applications ....................................................................... 23
  6.3.1 Image acquisition ............................................................... 23
  6.3.2 Image analysis .................................................................... 24
6.3.3 Image browser ................................................................. 24
6.4 CAMERA INITIALIZATION .................................................... 25
6.5 BUFFERED IMAGE ACQUISITION ....................................... 25
6.6 RAW IMAGE PROCESSING .................................................. 26
6.7 OTHER FUNCTIONS ............................................................ 26
6.8 DATA STRUCTURES ............................................................. 27
   6.8.1 Camera ........................................................................ 27
   6.8.2 Dataset ....................................................................... 27
   6.8.3 ImgAnalysisData ..................................................... 28
   6.8.4 ImgBrowserData .................................................... 29
   6.8.5 ImgManData .......................................................... 29
   6.8.6 IpOutput ................................................................. 30
   6.8.7 IpParam ................................................................. 30
   6.8.8 RawImg ................................................................. 32
7 CMLOG .................................................................................. 34
   7.1 STARTING CMLOG BROWSER ......................................... 34
   7.2 LOGGING MESSAGES .................................................. 34
8 MISCELLANEOUS FUNCTIONS ............................................... 35
   8.1 LCA2MATLABTIME .................................................... 35
   8.2 LCATS2PULSEID ....................................................... 35
9 AN EXAMPLE SESSION ........................................................ 36
APPENDIX A EPICS/SLC ATTRIBUTES ..................................... 43
   9.1 MAGNET ATTRIBUTES .................................................. 43
APPENDIX B AIDA TYPES ....................................................... 44
APPENDIX C MATLAB SUPPORT PVS ..................................... 45
0 Requirements

* A UNIX/AFS account
  http://www2.slac.stanford.edu/comp/slacwide/account/account.html
* A Red Hat 4 (or compatible linux) machine to log onto, e.g.
  facet-srv01.slac.stanford.edu
1 Basics

1.1 Setting up the environment

The FACET EPICS control system resides on its own MCC-based private network, parallel to and separate from the LCLS network. The FACET server that will be used for physics work is:

- facet-srv01

To log into the FACET network from a Linux terminal session:

- Account setup:
  - Your Unix account must be added to the FACET and fphysics groups. Contact Ken Brobeck (x2558).

- Login:
  - Bring up a Linux terminal window:
    - from an MCC OPI or Linux box click the terminal icon on the desktop
    - from Windows use Secure CRT or XWin-32
  - Log into mcclogin with your Unix account: ssh mcclogin
  - From mcclogin, log into facet-srv01 as the fphysics account ssh fphysics@facet-srv01
  - Enter the number corresponding to your username from the list. If you are not in the username list yet and would like be, then:
    - enter 0 (for None). You will end up in directory /home/fphysics.
    - mkdir username (username is your Unix login username)
    - logout (log out to reset the list)
    - Log back in: ssh fphysics@facet-srv01
    - Enter the number corresponding to your username
  - You should now be in /home/fphysics/username (e.g. /home/fphysics/fred)
  - Your environment should now be set up to run and develop MATLAB scripts.

- Optional: if you want to customize your environment further
  - Create file /home/fphysics/username/ENVS
  - ENVS will be sourced every time you log in

1.2 Starting MATLAB

Type

matlab

1.3 Directories
If you want to share your tested scripts, they will need to be released to the production directory, using CVS (see 1.4 CVS below)
/usr/local/facet/tools/matlab/toolbox

MATLAB scripts from LCLS/FACET software engineering team are in
/usr/local/facet/tools/matlab/src

1.4 CVS

You can use CVS to keep track of changes to your scripts, enable collaborative editing, compare to previous versions, etc.

1.4.1 Basic concept
CVS keeps up-to-date copies of your files in a single repository, and you along with other people work with copies of these files in a local directory. Think of CVS as a hyper-modern library where you can edit books. All shared matlab scripts are stored in the version control system, CVS. LCLS and FACET matlab scripts share a CVS repository, so there are many LCLS-specific scripts to be found in the toolbox and src directories, alongside the FACET and so-called "accelerator-agnostic" versions.
1.4.2 CVS Commands
All CVS commands follow the keyword `cvs`.

1.4.2.1 Checkout
To start working with CVS, you must check out the matlab directories.
```
>> cvs checkout matlab/toolbox
```

1.4.2.2 Commit
After you made changes to a local file (and tested it), you can commit it back to the repository.
```
>> cvs commit matlab/toolbox/moments.m
```
Don’t forget to enter a short, but comprehensive comment!

1.4.2.3 Update, cvs2prod
You can update your local directory with files from the repository, if you didn’t change their content yet.
```
>> cvs update
cvs update: Updating matlab
```
To update changes to production, i.e. make them generally available, you’ll need to use:
```
cvs2prod
```
For example you’re working on a script called myScript.m. In your working directory, enter:
```
cvs2prod myScript.m
```

1.4.2.4 Add
If you created a new local file, you can put it into the CVS repository.
```
>> cvs add matlab/toolbox/hist2.m
```

1.4.2.5 Remove
If you delete a local file, you can remove it from the CVS repository.
```
>> cvs remove matlab/toolbox/hist2.m
```

1.4.2.6 Diff
Sometimes you want to know how your local files differ from the ones in the repository.
>> cvs diff matlab/toolbox/hist2.m
1.4.2.7 Other commands
See for a comprehensive list of other CVS commands here
http://www.cvsnt.org/wiki/CvsCommand

1.4.3 CVSWEB
There is a website that gives you graphical access to the CVS repository
www.slac.stanford.edu/cgi-wrap/cvsweb/matlab/toolbox/?cvsroot=LCLS

1.4.4 Guidelines
Some good guidelines for using CVS can be found here
http://dotat.at/writing/cvs-guidelines.html
2 LabCA

Interface to EPICS, the FACET control system

The labCA toolbox wraps the essential ChannelAccess routines and makes them accessible from the MATLAB programs.

2.1 General concepts

2.1.1 Parameters and return values

All labCA calls take a PV argument identifying the EPICS process variable the user wants to access. EPICS PVs are plain ASCII strings that follow the pattern

<device>::<attribute>

LabCA is capable of handling multiple PVs in a single call; they are simply passed as a cell-array of strings, e.g.:

```matlab
pvs = { 'device:xyz'; 'PVa'; 'anotherone' }
```

2.1.2 Type

Unless all PVs are of native ‘string’ type or conversion to ‘string’ is enforced explicitly (type char), labCA always converts data to double. Legal values for type are byte, short, long, float, double, native, or char (for strings).

2.1.3 Timestamp format

ChannelAccess timestamps provide the number of nanoseconds since 00:00:00 UTC, January 1, 1970. LabCA translates the timestamp into a complex number with the seconds in the real and nanoseconds in the imaginary parts. To convert timestamps into MATLAB format use epics2matlabTime (see 8.1).

2.1.4 Exceptions

If a labCA command cannot execute correctly, it prints an error message and throws an exception. If the exception is not caught, the execution is aborted (look for details in the MATLAB manual).

```matlab
>> try
lcaGet('gibberish')
catch
'Reading from PV ''gibberish'' produced this error'
end
multi_ezca_get_nelem - ezcaGetNelem(): could not find process variable
```
2.1.5 Timeouts
Since labCA is used for accessing data via network, your function calls can timeout (see 2.4).

2.2 Basic commands
2.2.1 lcaGet

[value, timestamp] = lcaGet(pvs, nmax, type)

Description
Read a number of PVs.

Parameters
pvs
m x 1 cell- matrix of m strings.
nmax (optional)
Maximum number of elements (per PV) to retrieve (i.e. limit the number of
columns of value to nmax).
If set to 0 (default), all elements are fetched and the number of columns in the
result matrix is set to the maximum number of elements among the PVs.
This parameter is useful to limit the transfer time of large waveforms.
type (optional)
A string specifying the data type to be used for the channel access data transfer.

Returns
value
The m x n result matrix. n is automatically assigned to accommodate the PV with
the most elements.
Excess elements of PVs with less than n elements are filled with NaN values.
LabCA fills the rows corresponding to INVALID PVs with NaNs. In addition,
warning messages are printed to the console if a PV's alarm status exceeds a
configurable threshold.
timestamp
m x 1 column vector of complex numbers holding the CA timestamps of the
requested PVs.
The timestamps count the number of seconds (real part) and fractional
nanoseconds (imaginary part) elapsed since 00:00:00 UTC, Jan. 1, 1970.
Example

\[
\text{>> [values, timestamps] = lcaGet({'MIKE:BEAM'; 'MIKE:BEAM:RATE'}, 0, 'char')}
\]

\[
\text{values =}
\]

\[
'ON'
'1'
\]

\[
\text{timestamps =}
\]

\[
1.0e+09 *
\]

\[
1.1546 + 0.8052i
1.1546 + 0.5014i
\]

\[
\text{>> timestamps(1), timestamps(2)}
\]

\[
\text{ans =}
\]

\[
1.1546e+09 + 8.0522e+08i
\]

\[
\text{ans =}
\]

\[
1.1546e+09 + 5.0136e+08i
\]

2.2.2 lcaPut, lcaPutNoWait

\[
\text{lcaPut(pvs, value, type)}
\]
\[
\text{lcaPutNoWait(pvs, value, type)}
\]

\text{Description}
Write values to a number of PVs, which may be scalars or arrays of different dimensions.
It is possible to write the same value to a collection of PVs.
lcaPut will wait until the request is processed on the server, lcaPutNoWait returns immediately.

Parameters
pvs
m x 1 cell- matrix of m strings.
value
A matrix or a row array of values to be written to PVs. In latter case, the same value is written to all m PVs.
type (optional)
A string specifying the data type to be used for the channel access data transfer (see 2.1.2)

Example

>> lcaPut('MIKE:BEAM:RATE', 5)
>> lcaGet('MIKE:BEAM:RATE')

ans =

5

2.3 Monitors
Background
There is often a case where you need to wait until a PV value changes. While the code below works

```matlab
initialValue = lcaGet('waveformPV');
while(1)
    currentValue = lcaGet('waveformPV');
    if ~isequal(currentValue, initialValue)
        %do somet
        break;
    end
end
```

unnecessary network traffic can be the result if waveformPV stores big amount of data. To counter, labCA allows you to ask whether a PV value has changed. Then, you get the new value using lcaGet.
2.3.1 lcaSetMonitor

lcaSetMonitor(pvs, nmax, type)

Description
Set a "monitor" on a set of PVs. Use the lcaNewMonitorValue to check monitor status (local flag). If new data is available, it is retrieved using the lcaGet call. Use the lcaClear call to remove monitors on a channel.

Parameters
pvs
m x 1 cell- matrix of m strings.
nmax (optional)
Maximum number of elements (per PV) to monitor. If set to 0, all elements are fetched.
type (optional)
A string specifying the data type to be used for the channel access data transfer. The native type is used by default. The type specified for the subsequent lcaGet call should match the monitor's data type.

Example

>> lcaSetMonitor('MIKE:BEAM')

2.3.2 lcaNewMonitorValue

[flags] = lcaNewMonitorValue(pvs, type)

Description
Check if monitored PVs need to be read, i.e. if fresh data is available (due to initial connection or changes in value and/or severity status). Reading the actual data must be done using lcaGet.

Parameters
pvs
m x 1 cell- matrix of m strings.
type (optional)
A string specifying the data type to be used for the channel access data transfer. The native type is used by default. Note that monitors are specific to a particular data type and therefore lcaNewMonitorValue will only report the status for a monitor that had been established by lcaSetMonitor with a matching type.

Returns
flags
A cell-array of flag values. A value of 0 indicates that no new data is available - the monitored PV has not changed since it was last read. A value of 1 indicates that new data is available for reading (with `lcaGet`). A negative flag value indicates a problem:

-1: no monitor established (`lcaSetMonitor` never called for this PV/data type)
-2: non-existing PV (no successful CA search so far),
-3: invalid type argument,
-4: invalid PV argument,
-10: currently no connection.

Example

```matlab
>> lcaNewMonitorValue({'a';'b'})
ans =
    -2
    -2
```

### 2.3.3 lcaClear

*lcaClear*(pvs)

Description

Disconnect PVs. All monitors on the target channel(s) are cancelled/released as a consequence of this call.

Parameters

pvs (optional)

m x 1 cell- matrix of m strings. If omitted, disconnects all PVs.

Example

```matlab
>> lcaClear
```

### 2.4 Network-related settings

The default labCA timeout for ChannelAccess calls is 0.1s and the default number of retries is 299. This means that if something does not work on the server, the labCA
commands will wait for up to 29.9s before they return. We recommend changing these values.

2.4.1 Timeout

currentTimeout = lcaGetTimeout()

Description
Retrieve current timeout (in seconds).

Returns
currentTimeout
Current timeout in seconds

lcaSetTimeout(newTimeout)

Description
Set the new timeout.

Parameters
newTimeout
New timeout in seconds.

Example

>> lcaSetTimeout(0.1)
>> lcaGetTimeout

ans =

    0.1000

2.4.2 Retry count

currentRetryCount = lcaGetRetryCount()

Description
Retrieve the number of retries.

Returns
currentRetryCount
lcaSetRetryCount(newRetryCount)

Description
Set the number of retries.

Parameters
newRetryCount

Example

>> lcaSetRetryCount(10)
>> lcaGetRetryCount

ans =

10

2.4.3 Default configuration

You may want to choose to include the default FACET labCA configuration script lcaInit in your startup.m file. See MATLAB documentation for details.

2.5 Other
For all supported labCA commands see http://www.slac.stanford.edu/~strauman/labca/manual/node2.html
3 AIDA
Interface to SLC model data

SLC model data (e.g. TWISS parameters) is not accessible via ChannelAccess. Instead, you must use the Java-based library called AIDA (Accelerator Integrated Data Access).

3.1 Java setup
Make sure you have a java.opts file in the directory from which you start MATLAB.

3.2 Browsing names

You can browse AIDA names by typing

aidalist(device, attribute)

As a temporary solution, this function will prompt for your password.

Parameters
device
   AIDA device name
attribute (optional)
   AIDA attribute name

Note: As a wild character, you can use the % character (but NOT *). However, in the device parameter you must at least include one other character (also, be mindful of the output).

Returns
The output is a list of names known to AIDA.

Example

>> aidalist(‘QUAD:IM20:221’, ‘Z%’)

chevtsov@flora’s password:

QUAD:IM20:221
QUAD:IM20:221

ans =

0

3.3 Retrieving data

To get data, use the function

aidaget(aida_name, type, params)

Parameters
aida_name

AIDA name following the pattern

<device>//<attribute>

Note: Device/attribute names used by AIDA differ from the ones used by EPICS, see Appendix A.

type

(optional) case-insensitive string. Choose type from the table in Appendix C.

params

(optional) a cell array of AIDA parameters. Use the “=” notation for each parameter and value.

Returns

SLC value(s) of the specified type

Example 1

>> aidaget('YCOR:PR12:1052//BACT', 'double')

Fri Sep 15 11:58:48 PDT 2006: Making connection to Name Service
Fri Sep 15 11:58:48 PDT 2006: Making connection to daServer

ans =

0

Example 2

>> r = aidaget('BPMS:IA20:221//R', 'doublea', {'B=BPMS:IA20:511'});

Fri Sep 15 12:00:06 PDT 2006: Making connection to Name Service
Fri Sep 15 12:00:06 PDT 2006: Making connection to daServer

>> rMatrix = reshape(r, 6, 6)

rMatrix =

Columns 1 through 3

[-0.7087]    [  0.1886]  [-7.0392e-19]
[-0.1078]    [  0.1703]  [-9.7456e-20]
[  1.6913e-19]  [4.6419e-19]  [-2.5244e-29]
[-2.5244e-29]  [  6.2426e-20]  [-3.9345e-30]
[  0]     [  1.5362e-19]  [0]
[ 0]
Columns 4 through 6

\[
\begin{bmatrix}
1.8859 \times 10^{-19} & 0 & -2.4379 \times 10^{-32} \\
-1.0779 \times 10^{-19} & 0 & 1.0095 \times 10^{-32} \\
0.0975 & 0 & 0 \\
-0.0624 & 0 & 0 \\
0 & 1 & -3.2001 \times 10^{-28} \\
0 & 0 & 0.0443 \\
\end{bmatrix}
\]

3.4 Miscellaneous

For more information like possible parameters, please see http://www.slac.stanford.edu/grp/cd/soft/aida
4 Beam synchronous acquisition (BSA)

A FACET event system has been setup to read devices synchronous with beam crossing. An interface between FACET EPICS device BSA and SCP device BSA is under development. Details coming... What follows is the description for LCLS, some of which may apply for FACET as well.

This system can be used from within Matlab with a few simple calls. Note that this is not implemented for image data collection. See separate section on collecting image data and collecting image data along with other beam synchronous devices. Figure 4.1 shows how the IOC looks at the EVR signal and BPM data to fill its BSA buffers.

![Conceptual data flow for Beam Synchronous Acquisition](image)

Figure 1: Conceptual data flow for Beam Synchronous Acquisition

4.1 Reserving an event definition

eDefNumber = eDefReserve(myName)
Description
Reserves an event definition for your use.

Parameters
myName
   a unique name for your application

Returns
ID of an event definition

4.2 Changing default parameters

eDefParams (eDefNumber, navg, nrpos, incmSet, incmReset, excmSet, excmReset )

Description
Changes defaults of the supplied arguments

Parameters
eDefNumber
   ID of an event definition
navg
   Number of pulses to average per reading To reduce the amount of jitter, you may choose to average several pulses together to get an averaged value. Default is no averaging; maximum is currently 10.
nrpos
   Number of readings to take, which is the buffer size you wish to acquire. Default is one reading; maximum is currently 2800.
incmSet, incmReset, excmSet, excmReset
   Optional Inclusion and Exclusion Timing Modifiers. Specific PNET bits you wish to be present or absent during your acquisition, such as ONE_HERTZ. See the PNBN database for a list of available PNET bits by viewing the PVs MP00:PNBN: [1..150]:NAME. Defaults to the values in VX00:DGRP:1150:INCM and VX00:DGRP:1150:EXCM.

4.3 Starting data acquisition

elapsedTime = eDefAcq (eDefNumber, timeout)

Description
Starts the data acquisition cycle. If the elapsed time is less than the timeout specified then your buffers are completely populated with the number of readings you specified, otherwise your buffers are populated with the number of readings the system was able to collect in the time allotted for data acquisition. This function blocks Matlab execution. Please see the script for a non-blocking example.
You collect your data from PVs with \texttt{lcaGet}. If, for example, you're interested in the device BPMS:IN20:221, your data can be found in the following PVs:

BPMS:IN20:221:XHST\texttt{[eDefNumber]}
Waveform containing all BPMS X values collected

BPMS:IN20:22:1:X\texttt{[eDefNumber]}
The last BPMS X value collected, handy when you only requested one reading

BPMS:IN20:221:X\texttt{[eDefNumber]}\texttt{.H}
Rms of the last BPMS X value collected

BPMS:IN20:221:MEASCNT\texttt{[eDefNumber]}
Number of beam pulses used in your acquisition

Data for all devices known to the event definition system is available. There is no need to specify a device list for data acquisition. For an explanation of all available devices, please see the documentation on event definitions. For a complete list of BSA pvs, please try the Aida query FACET/* at \url{https://seal.slac.stanford.edu/aidaweb}

4.4 Releasing an event definition

\texttt{eDefRelease (eDefNumber)}

\textbf{Description}
Releases your event definition for use by another control system application. Note that you can perform many data acquisition and collection cycles before releasing your event definition reservation.

\textbf{Parameters}
\texttt{eDefNumber}
ID of an event definition

4.5 An example script

The example can be found in eDefExample.m

% Choose unique name
myName = 'Matlab eDef';
myNAVG = 1;
myNRPOS = 20;
timeout = 3.0; % seconds

% Reserve an eDef number
myeDefNumber = eDefReserve(myName);

% Make sure I got an eDef Number
if isequal (myeDefNumber, 0)
    disp ('Sorry, failed to get eDef');
else
    disp (sprintf('%s I am eDef number %d',datestr(now),myeDefNumber));
end

% set my number of pulses to average, etc... Optional, defaults to no
% averaging with one pulse and DGRP INCM & EXCM.
eDefParams (myeDefNumber, myNAVG, myNRPOS, {'TS4'},{''},{'TS1'},{''});

acqTime = eDefAcq(myeDefNumber, timeout);
if (acqTime < timeout)
    disp (sprintf ('%s Data collection complete, took %.1f seconds',
        datestr(now), acqTime));
else
    disp (sprintf ('%s Data collection timed out. Data available for
        %.1f seconds', datestr(now), acqTime));
end

% read data, note that data stays until you give up your eDef
xVec = lcaGet(sprintf('BPMS:IN20:221:XHST%d',myeDefNumber));
yVec = lcaGet(sprintf('BPMS:IN20:221:YHST%d',myeDefNumber));
iVec = lcaGet(sprintf('BPMS:IN20:221:TMITHST%d',myeDefNumber));
pidVec = lcaGet(sprintf('PATT:SYS0:1:PULSEIDHST%d',myeDefNumber));

disp(sprintf('Event definition (EVG) claimed to have collected %d
   steps', eDefCount(myeDefNumber));

% Give up eDef
eDefRelease(myeDefNumber);
6 Image management

The image management (ImgMan) toolbox is a set of Matlab functions and GUls for online and off-line processing of grayscale radiation images resulting from impact of electrons on a screen in the beam line. The toolbox includes applications for acquiring, browsing, and analyzing images.

6.1 Overview

ADD PICTURE!!! (Coming April 12th, 2008)

6.2 Global data

ImgMan accesses raw image data and makes available processed images and beam data through a global variable called gIMG_MAN_DATA, which is an instance of the ImgManData struct (see 6.8.5).

6.3 GUI applications

6.3.1 Image acquisition

\[ h = \text{imgAcq\_main(cameraArray)} \]

Description
Starts the image acquisition application.

Parameters
cameraArray
A row cell array of camera structs (see 6.8.1).

Returns
The handle of the main image acquisition figure.

Example

\[ \text{imgAcq\_main();} \]
6.3.2 Image analysis

\[ h = \text{imgAnalysis}\_main(\text{imgAnalysisData}, \text{left}, \text{top}) \]

Description

Starts the image analysis application.

Parameters

- **imgAnalysisData** (optional)
  - An instance of the `imgAnalysisData` struct (see 6.8.3).
- **left** (optional)
  - X coordinate of the upper left corner of the figure.
- **top** (optional)
  - Y coordinate of the upper left corner of the figure.

Returns

The handle of the main image analysis figure.

Example

```matlab
h = imgAnalysis\_main();
set(h, 'name', 'My Image Analysis');
```

6.3.3 Image browser

\[ h = \text{imgBrowser}\_main(\text{imgBrowserData}, \text{left}, \text{top}) \]

Description

Starts the image browser application. Note: image browser displays valid datasets only.

Parameters

- **imgBrowserData** (optional)
  - An instance of the `imgBrowserData` struct (see 6.8.4).
- **left** (optional)
  - X coordinate of the upper left corner of the figure.
- **top** (optional)
  - Y coordinate of the upper left corner of the figure.

Returns

The handle of the main image browser figure.

Example

```matlab
h = imgBrowser\_main();
set(h, 'name', 'My Browser');
```
6.4 Camera initialization

cameraArray = imgAcq_initCameraProperties()

Description
Initializes the properties of all available cameras. This function is designed to be edited as if it was a properties file.

Returns
A row cell array of camera structs (see 6.8.1).

Example

TO DO

6.5 Buffered image acquisition

rawImgArray = imgAcq_runBufferedAcq(camera, nrBgImgs, nrBeamImgs, saveBufferedImgs, progHandles)

Description
Executes the buffered image acquisition and returns a list of images, sorted by timestamps. Background images are retrieved before beam images. If you request 0 background images, a saved background image will be retrieved first.

Parameters

camera
An instance of the camera struct (see 6.8.1).
nrBgImgs
Number of background images (>=0).
nrBeamImgs
Number of beam images (>=0).
saveBufferedImgs (optional)
A flag indicating whether buffered images should be saved locally. Default value is 1.
progHandles (optional)
Handles of the GUI that contains a progress panel. Default value is [].

Returns
A row cell array of rawImg structs (see 6.8.8).

Example
6.6 Raw image processing

\[
ipOutput = \text{imgProcessing\_processRawImg}(\text{rawImg}, \text{camera}, \text{ipParam}, \text{bgImg})
\]

Description
Processes raw image data from the specified camera according to the specified image processing parameters.

Parameters
- rawImg
  An instance of the rawImg struct (see 5.8.8).
- camera
  An instance of the camera struct (see 5.8.1).
- ipParam
  An instance of ipParam struct (see 5.8.7).
- bgImg (optional)
  A grayscale image that represents the background noise.

Returns
An instance of the ipOutput struct (see 5.8.6).

Example

TO DO

6.7 Other functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imgAcq_epics*</td>
<td>A set of functions for low(er)-level interactions with EPICS.</td>
</tr>
<tr>
<td>imgData_construct*</td>
<td>A set of functions for creating default</td>
</tr>
</tbody>
</table>
6.8 Data structures

6.8.1 Camera
Default constructor: `imgData_construct_camera`

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Default value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bufferSize</td>
<td>Size of the buffer for images this camera can take.</td>
<td>0</td>
<td>Integer</td>
</tr>
<tr>
<td>hasScreen</td>
<td>Flag indicating whether this camera points at a screen.</td>
<td>0</td>
<td>Boolean</td>
</tr>
<tr>
<td>isProd</td>
<td>A flag indicating whether this camera is on production network.</td>
<td>1</td>
<td>Boolean</td>
</tr>
<tr>
<td>label</td>
<td>The label of the screen.</td>
<td>‘N/A’</td>
<td>String.</td>
</tr>
<tr>
<td>maxNrBeamImgs</td>
<td>Due to high costs of the screens, we decided to limit the number of beam images that users can request per measurement.</td>
<td>0</td>
<td>Integer.</td>
</tr>
<tr>
<td>pvPrefix</td>
<td>Prefix for PVs that deliver other camera parameters.</td>
<td>‘N/A’</td>
<td>String.</td>
</tr>
<tr>
<td>updatePause</td>
<td>Time between requests for live images from this camera.</td>
<td>0</td>
<td>Double (&gt;= 0).</td>
</tr>
</tbody>
</table>

6.8.2 Dataset
Default constructor: `imgData_construct_dataset`

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Default value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>camera</td>
<td>The camera which datasets was taken from.</td>
<td>See 5.8.1.</td>
<td>Camera struct.</td>
</tr>
<tr>
<td>ipOutput</td>
<td>A list of image processing output parameters for external applications. Note: ImgMan does not read it.</td>
<td>[] (empty)</td>
<td>Row cell array of ipOutput structs (see 5.8.6).</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Default value</td>
<td>Type</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>------------</td>
</tr>
<tr>
<td>ipParam</td>
<td>A set of image processing parameters for external applications. Note: ImgMan does not read this field.</td>
<td>[] (empty)</td>
<td>IpParam struct (see 5.8.7).</td>
</tr>
<tr>
<td>isValid</td>
<td>A flag indicating the validity of the dataset. ImgBrowser does not show invalid datasets.</td>
<td>1</td>
<td>Boolean.</td>
</tr>
<tr>
<td>label</td>
<td>The label of this dataset.</td>
<td>‘n/a’</td>
<td>String.</td>
</tr>
<tr>
<td>masterCropArea</td>
<td>The area of the master crop region.</td>
<td>[]</td>
<td>Four-element row array of doubles, [xmin ymin width height] each in default spatial coordinates.</td>
</tr>
<tr>
<td>nrBgImgs</td>
<td>Number of background images in the dataset. ImgMan treats the first nrBgImgs images in the dataset as background images.</td>
<td>0</td>
<td>Integer.</td>
</tr>
<tr>
<td>nrBeamImgs</td>
<td>Number of beam images in the dataset.</td>
<td>0</td>
<td>Integer.</td>
</tr>
<tr>
<td>rawImg</td>
<td>A list of raw image parameters.</td>
<td>[] (empty)</td>
<td>Row cell array of rawImg structs (see 5.8.8).</td>
</tr>
</tbody>
</table>

### 6.8.3 ImgAnalysisData
Default constructor: `imgData_construct_imgAnalysis`

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Default value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsIndex</td>
<td>The index of a dataset (which doesn’t have to be valid).</td>
<td>1</td>
<td>Integer.</td>
</tr>
<tr>
<td>imgIndex</td>
<td>The index of an image in the selected dataset.</td>
<td>1</td>
<td>Integer.</td>
</tr>
</tbody>
</table>
### 6.8.4 ImgBrowserData

Default constructor: `imgData_construct_imgBrowser`

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Default value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>fitPlane</td>
<td>The plane of the initial fits.</td>
<td>‘x’</td>
<td>‘x’ or ‘y’</td>
</tr>
<tr>
<td>imgOffset</td>
<td>The offset of the displayed images.</td>
<td>0</td>
<td>Integer.</td>
</tr>
<tr>
<td>ipOutput</td>
<td>Initial set of image processing output parameters.</td>
<td>[] (empty)</td>
<td>IpOutput struct (see 5.8.6).</td>
</tr>
<tr>
<td>ipParam</td>
<td>Initial set of image processing parameters.</td>
<td>[] (empty)</td>
<td>IpParam struct (see 5.8.7).</td>
</tr>
<tr>
<td>nrDsTabs</td>
<td>The number of visible dataset tabs in the ImgBrowser figure. You should adjust it if you have datasets with long labels.</td>
<td>5</td>
<td>Integer.</td>
</tr>
<tr>
<td>validDsIndex</td>
<td>The index of the selected valid dataset. ImgMan uses its own defaults, if you provide a negative value.</td>
<td>-1</td>
<td>Integer.</td>
</tr>
<tr>
<td>validDsOffset</td>
<td>The offset of the displayed datasets. ImgMan uses its own defaults, if you provide a negative value.</td>
<td>-1</td>
<td>Integer.</td>
</tr>
</tbody>
</table>

### 6.8.5 ImgManData

Default constructor: `imgData_construct_imgMan`
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Default value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>A list of raw image datasets.</td>
<td>[] (empty)</td>
<td>Row cell array of dataset structs (see 5.8.2).</td>
</tr>
<tr>
<td>ipOutputChanged</td>
<td>A flag indicating whether new processed images and/or beam data is available. Note: ImgMan sets this field only to 1; it is up to the client to reset it to 0.</td>
<td>0</td>
<td>Boolean.</td>
</tr>
<tr>
<td>isDirty</td>
<td>A flag indicating whether the datasets need to be saved.</td>
<td>0</td>
<td>Boolean.</td>
</tr>
</tbody>
</table>

### 6.8.6 IpOutput

Default constructor: `imgData_construct_ipOutput`

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Default value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>beamlist</td>
<td>A set of beam fit parameters.</td>
<td>[] (empty)</td>
<td>TO DO</td>
</tr>
<tr>
<td>isValid</td>
<td>The flag indicating whether the raw image is valid according to the image processing algorithm.</td>
<td>[]</td>
<td>Boolean.</td>
</tr>
<tr>
<td>offset</td>
<td>Pixel coordinates of the upper left corner of the processed image (useful e.g. if the raw image was cropped).</td>
<td>0 and 0</td>
<td>Double.</td>
</tr>
<tr>
<td>.x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>procImg</td>
<td>The result of image processing.</td>
<td>0</td>
<td>A matrix of doubles.</td>
</tr>
</tbody>
</table>

### 6.8.7 IpParam

Default constructor: `imgData_construct_ipParam`
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>algIndex</td>
<td>ImgMan’s image processing routine returns sets of beam data for each of the pre-defined algorithms in bulk. This field is used to display data that resulted from an algorithm with the particular index.</td>
<td>Integer.</td>
</tr>
<tr>
<td>annotation.centroid.current.color</td>
<td>Color of the current beam centroid mark.</td>
<td>[0 0 0] (black) RGB color.</td>
</tr>
<tr>
<td>annotation.centroid.current.flag</td>
<td>Flag indicating whether to display the current beam centroid mark.</td>
<td>0 Boolean.</td>
</tr>
<tr>
<td>annotation.centroid.goldenOrbit.color</td>
<td>Color of the golden orbit beam centroid mark.</td>
<td>[1 0.84 0] (gold) RGB color.</td>
</tr>
<tr>
<td>annotation.centroid.goldenOrbit.flag</td>
<td>Flag indicating whether to display the golden orbit beam centroid mark.</td>
<td>0 Boolean.</td>
</tr>
<tr>
<td>annotation.centroid.laserBeam.color</td>
<td>Color of the laser beam centroid mark.</td>
<td>[1 0 0] (red) RGB color.</td>
</tr>
<tr>
<td>annotation.centroid.laserBeam.flag</td>
<td>Flag indicating whether to display the laser beam centroid mark.</td>
<td>0 Boolean.</td>
</tr>
<tr>
<td>beamSizeScaleUnit</td>
<td>The scale unit of the beam size.</td>
<td>‘pix’ ‘pix’, or ‘mm’</td>
</tr>
<tr>
<td>colormapFcn</td>
<td>Name of the function that creates a colormap.</td>
<td>‘jet’ String</td>
</tr>
<tr>
<td>crop.auto.custom</td>
<td>Flags indicating whether an automatic crop by the image processing algorithms or a custom crop is desired. The flags should not be set to 1 at the same time.</td>
<td>Auto = 0 Custom = 0 Boolean.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Default value</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Floor .floor .median</td>
<td>Flags indicating whether a filter should be applied to the raw image.</td>
<td>Floor = 0 Median = 0</td>
</tr>
<tr>
<td>lineWidthFactor</td>
<td>A factor for the width of the annotation lines (in proportion to the size of the corresponding axes).</td>
<td>1/150</td>
</tr>
<tr>
<td>slice .index .plane .total</td>
<td>The current index, plane, and total number of slices of the raw image. ImgMan extracts beam data from the specified slice only.</td>
<td>Index = 1 Plane = ‘x’ Total = 1</td>
</tr>
<tr>
<td>nrColors .max .min .val</td>
<td>Maximum, minimum, and the current number of colors in the colormap.</td>
<td>Max = 4084; Min = 64; Val = 256;</td>
</tr>
<tr>
<td>subtractBg .acquired .calculated</td>
<td>Flags indicating whether to subtract the background noise that is calculated by the image processing algorithms, or the average of the background images in the dataset from raw images. If both flags are set to 0, no image processing is done. The flags should not be set to 1 at the same time.</td>
<td>Acquired = 0 Calculated = 0</td>
</tr>
</tbody>
</table>

### 6.8.8 RawImg

Default constructor: `imgData_construct_rawImg`
<table>
<thead>
<tr>
<th>customCropArea</th>
<th>Coordinates of a custom region of interest for the raw image.</th>
<th>[] (empty)</th>
<th>Four-element row array of doubles, [xmin ymin width height] each in default spatial coordinates.</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Grayscale image data that ImgMan retrieves from an IOC.</td>
<td>0</td>
<td>Matrix of integers.</td>
</tr>
<tr>
<td>ignore</td>
<td>A flag for ignoring the raw image, overriding the validity field of an instance of IpOutput (see 6.8.6).</td>
<td>[]</td>
<td>Boolean or [].</td>
</tr>
<tr>
<td>timestamp</td>
<td>The time of when the camera delivered the raw image to the IOC.</td>
<td>-1</td>
<td>LabCA timestamp.</td>
</tr>
</tbody>
</table>
7 Cmlog
Control system logging facility

All commands can be executed outside MATLAB. Use -help option to get more information about each command.

7.1 Starting cmlog browser

To start cmlog from MATLAB, type

unix('cmlog -c -u &')

Note: Don’t forget & or your MATLAB will block!

7.2 Logging messages

To log a message to cmlog, type

unix('cmlogMsgLine ''some message''')
8 Miscellaneous functions

This chapter describes useful MATLAB functions developed at LCLS.

8.1 lca2matlabTime

\texttt{lca2matlabTime(lcaTS)}

Converts labCA timestamp into MATLAB time (number of days since 1/1 0000).

Parameter

\texttt{lcaTS}

LabCA timestamp as acquired through \texttt{lcaGet}.

Example

\begin{verbatim}
>> [value, lcaTS] = lcaGet('MIKE:BEAM');
>> matlabTS = lca2matlabTime(lcaTS);
>> datestr(matlabTS, 'mmm dd HH:MM:SS.FFF')
\end{verbatim}

\texttt{ans =}

Aug 23 18:04:18.658

8.2 lcaTs2PulseId

\texttt{lcaTs2PulseId(lcaTS)}

Extracts pulse ID from labCA timestamp

Parameter

\texttt{lcaTS}

LabCA timestamp as acquired through \texttt{lcaGet}.

Example

\begin{verbatim}
>> [value, lcaTS] = lcaGet('MIKE:BEAM');
>> lcaTs2PulseId(lcaTS);
>> ans =
\end{verbatim}

16753
9 An example session

Log onto a Linux computer

dhcpvisitor21831:~ chevtsov$ ssh noric05

Start MATLAB

chevtsov@noric05 $ matlab -nodesktop -nosplash

Get the status of the (fake) OTR 2 screen

>> lcaGet('MIKE:OTR2:POS')

ans =

    'OUT'

Put OTR2 screen in

>> lcaPut('MIKE:OTR2:POS:REQ', 'IN')

Get image from OTR2

>> epicsImg = lcaGet('MIKE:OTR2:IMAGE');

Reshape into a 2D image

>> img = reshape(img, 240, 320);
Display image

Result
## Appendix A  EPICS/SLC attributes

### 9.1 Magnet attributes

(Note: some might not be available yet)

<table>
<thead>
<tr>
<th>EPICS</th>
<th>SLC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACT</td>
<td>BACT</td>
<td>Physical Units val converted from IACT</td>
</tr>
<tr>
<td>BACT.ADEL</td>
<td>TOLS3</td>
<td>Alarm Delta</td>
</tr>
<tr>
<td>BCTRL</td>
<td>-</td>
<td>Set B field value, then perturb; updates BDES</td>
</tr>
<tr>
<td>BCTRL.DRVH</td>
<td>BMAX</td>
<td>Maximum Desired Value, B field</td>
</tr>
<tr>
<td>BDELTAS.B</td>
<td>TOLS(1)</td>
<td>Warning Delta</td>
</tr>
<tr>
<td>BDELTAS.C</td>
<td>TOLS(2)</td>
<td>Warning Fraction</td>
</tr>
<tr>
<td>BDELTAS.E</td>
<td>TOLS(3)</td>
<td>Alarm Fraction</td>
</tr>
<tr>
<td>BDELTAS.F</td>
<td>TOLS(4)</td>
<td>Monitor Fraction</td>
</tr>
<tr>
<td>BDES</td>
<td>BDES</td>
<td>Desired B field Value</td>
</tr>
<tr>
<td>BMON</td>
<td>BMON</td>
<td>Physical Units val converted from IMON</td>
</tr>
<tr>
<td>CALBOK</td>
<td>-</td>
<td>1=Calibration successful</td>
</tr>
<tr>
<td>CALBTOD.TIME</td>
<td>CTIM</td>
<td>Last successful calibration time</td>
</tr>
<tr>
<td>CTRL</td>
<td>-</td>
<td>Control Request \u2013 magnet function</td>
</tr>
<tr>
<td>FBCKCTRL</td>
<td>-</td>
<td>1=Magnet under feedback control</td>
</tr>
<tr>
<td>HSTA</td>
<td>HSTA</td>
<td>Translates SLC HSTA to individual EPICS records</td>
</tr>
<tr>
<td>HSTACTRL</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>IACT</td>
<td>IACT</td>
<td>Control Value from Hardware (Primary)</td>
</tr>
<tr>
<td>IACT.ADEL</td>
<td>ATOL(1)</td>
<td>Standardization Delta</td>
</tr>
<tr>
<td>IACT.HIHI, LOLO</td>
<td>IMMS</td>
<td>Standardization, Calib Control Limits</td>
</tr>
<tr>
<td>IACTPREVOK</td>
<td>IPRV</td>
<td>Last Good Readback</td>
</tr>
<tr>
<td>ICTRL.DRVH,DRVL</td>
<td>IMMO</td>
<td>Control Limits</td>
</tr>
<tr>
<td>IDAC</td>
<td>DACV</td>
<td>DAC Readback</td>
</tr>
<tr>
<td>IDELTAS.C</td>
<td>ATOL(2)</td>
<td>Standardization Fraction</td>
</tr>
<tr>
<td>IDES</td>
<td>IDES</td>
<td>Desired Control Value (amperes)</td>
</tr>
<tr>
<td>IMON</td>
<td>IMON</td>
<td>Control Value from Hardware (Secondary)</td>
</tr>
<tr>
<td>INSERVICE</td>
<td>-</td>
<td>1=in service</td>
</tr>
<tr>
<td>IRIPL</td>
<td>RIPL</td>
<td>Ripple Current Readback</td>
</tr>
<tr>
<td>NOCTRL</td>
<td>-</td>
<td>1=Can’t control this magnet; 0 otherwise</td>
</tr>
<tr>
<td>POLYCOEF.A to L</td>
<td>IVBU</td>
<td>Polynomial Coefficients</td>
</tr>
<tr>
<td>POLYCOEF.A to L</td>
<td>IVBD</td>
<td>Same as IVBU</td>
</tr>
<tr>
<td>STAT</td>
<td>STAT</td>
<td>Status Bits</td>
</tr>
<tr>
<td>STATUS</td>
<td>-</td>
<td>Status of Control Action</td>
</tr>
<tr>
<td>STDZDIRECTION</td>
<td>-</td>
<td>UP/DOWN direction of standardization</td>
</tr>
<tr>
<td>STDZENABLE</td>
<td>-</td>
<td>1=Standardize enabled</td>
</tr>
<tr>
<td>STDZIMAX or IDES.HIGH</td>
<td>IMAX</td>
<td>Max or min current during standardization</td>
</tr>
<tr>
<td>STDZLOSTTOD.TIME</td>
<td>ZTIM</td>
<td>Last time standardization lost</td>
</tr>
<tr>
<td>STDZTOD.TIME</td>
<td>KTIM</td>
<td>Last successful standardization time</td>
</tr>
<tr>
<td>TRIMENABLE</td>
<td>-</td>
<td>1=Trim enabled</td>
</tr>
<tr>
<td>TRIMTOD.TIME</td>
<td>TTIM</td>
<td>Last trim time</td>
</tr>
<tr>
<td>TRIMTRYCNT</td>
<td>ITRY</td>
<td># of attempts required for trim</td>
</tr>
</tbody>
</table>
Appendix B  AIDA types

Note: xxxA means array of type ‘xxx’

BOOLEAN
BOOLEANA
BYTE
BYTEA
CHAR
CHARA
DOUBLE
DOUBLEA
FLOAT
FLOATA
LONG
LONGA
LONGDOUBLE
LONGDOUBLEA
LONGLONG
LONGLONGA
SHORT
SHORTA
STRING
STRINGA
ULONG
ULONGA
ULONGLONG
ULONGLONGA
USHORT
USHORTA
WCHAR
WCHARA
WSTRING
WSTRINGA
Appendix C  Matlab support PVs

A set of Matlab support PVs for FACET are available from the matlab support soft IOCs, with the naming convention

SIOC:SYS1:ML0x:<type><n>

x is the IOC number, from 1 - 3

- type indicates PV type
- n is a 3-digit number

To view the PVs:
- bring up facethome
- click the “Matlab GUIs…” button
- click the “Matlab PVs…” button (bottom center)
- click the button for the PV type you’re interested in (e.g. “Matlab Support PVs…”), which are analog output records)
- in the dropdown list, select the range of PVs to view

To use the PVs:
- Use the edm displays above to find PV(s) that are not used already
- Edit in the edm display to replace “comment” with your usage.
- Use PV name(s) in your matlab program