MATLAB Programmer's Guide for FACET physicists

(Version 0.81)

LCLS version: 9/22/08

small changes for FACET April, 2011 *jrock*

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 environmen

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 or
    - 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 customized 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.:

```
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).

```
>> try
lcaGet('gibberish')
catch
'Reading from PV ''gibberish'' produced this error'
end
multi_ezca_get_nelem -  ezcaGetNelem(): could not find process variable
```

```
: gibberishError: Errors encountered...

ans =

Reading from PV 'gibberish' produced this error
```

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

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

values =

    'ON'
    '1'


timestamps =

   1.0e+09 *

   1.1546 + 0.8052i
   1.1546 + 0.5014i


>> timestamps(1), timestamps(2)

ans =

   1.1546e+09 + 8.0522e+08i


ans =

   1.1546e+09 + 5.0136e+08i
```

2.2.2 lcaPut, lcaPutNoWait

```
lcaPut(pvs, value, type)
lcaPutNoWait(pvs, value, type)
```

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

```
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

```
>> 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

```
>> 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                                                    Z
QUAD:IM20:221                                                 ZTIM

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]
   [     0.1703]    [    -0.1078]    [ 1.6913e-19]
   [ 4.6419e-19]    [-9.7456e-20]    [    -0.4677]
   [ 1.5362e-19]    [ 6.2426e-20]    [    -0.1548]
   [-2.5244e-29]    [-3.9345e-30]    [          0]
```

```
  [          0]    [          0]    [          0]

 Columns 4 through 6

  [ 1.8859e-19]    [0]    [-2.4379e-32]
  [-1.0779e-19]    [0]    [ 1.0095e-32]
  [     0.0975]    [0]    [          0]
  [    -0.0624]    [0]    [          0]
  [          0]    [1]    [-3.2001e-28]
  [          0]    [0]    [     0.0443]
```

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.

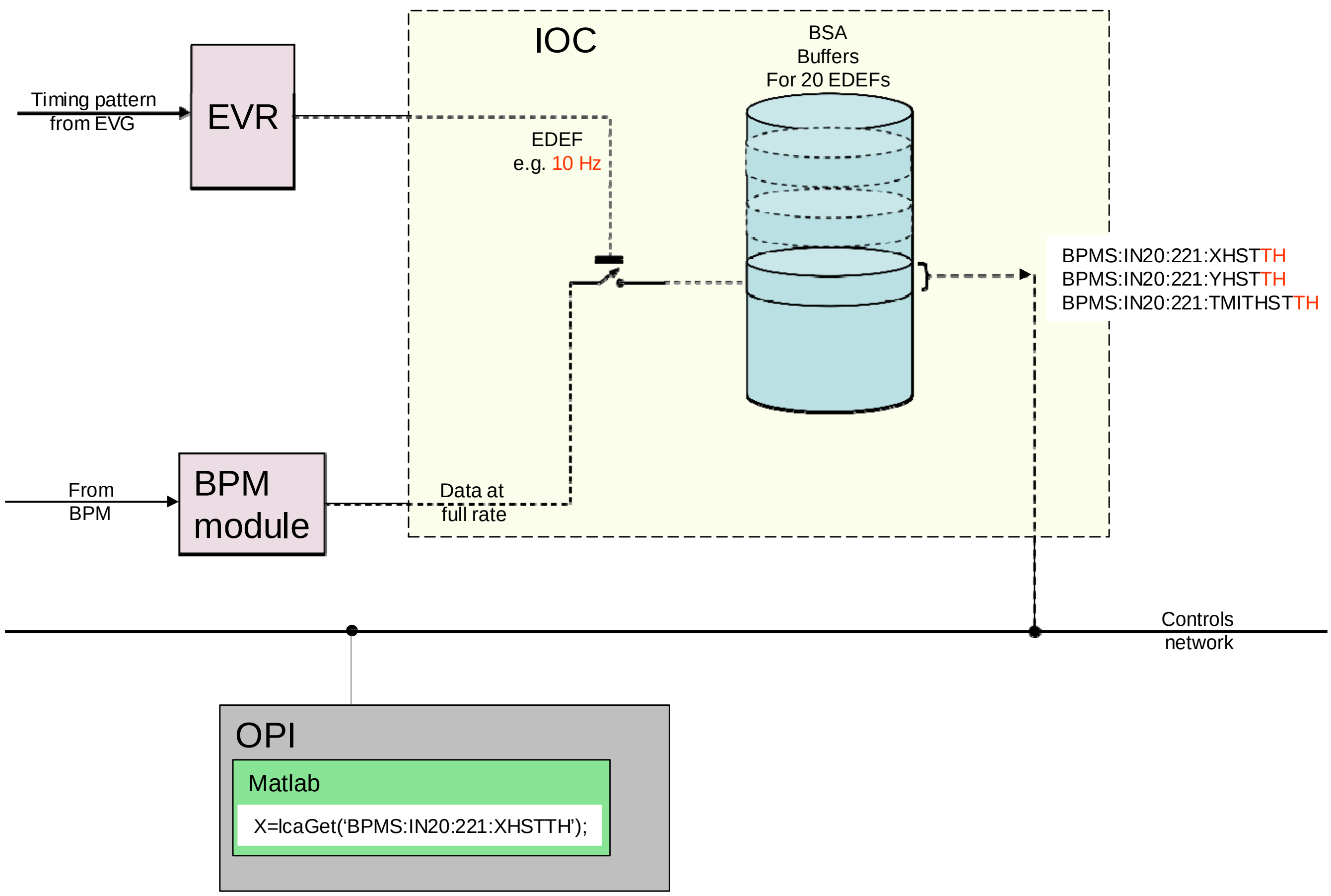


Figure 1: Conceptual data flow for Beam Synchronous Acquisition

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 `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[`eDefNumber`]
Waveform containing all BPMS X values collected

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

BPMS:IN20:221:X[`eDefNumber`].H
Rms of the last BPMS X value collected

BPMS:IN20:221:MEASCNT[`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 en 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 https://seal.slac.stanford.edu/aidaweb

4.4 Releasing an event definition

```
eDefRelease (eDefNumber)
```

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.

Parameters
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));

% 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 GUIs 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 = 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

```
imgAcq_main();
```

6.3.2 Image analysis

```
h = imgAnalysis_main(imgAnalysisData, left, 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

```
h = imgAnalysis_main();
set(h, ‘name’, ‘My Image Analysis’);
```

6.3.3 Image browser

```
h = imgBrowser_main(imgBrowserData, left, 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

```
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

```
TODO
```

6.6 Raw image processing

```
ipOutput = imgProcessing_processRawImg(rawImg, camera, ipParam, 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

Name	Description
imgAcq_epics*	A set of functions for low(er)-level interactions with EPICS.
imgData_construct*	A set of functions for creating default

6.8 Data structures

6.8.1 Camera

Default constructor: `imgData_construct_camera`

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

6.8.2 Dataset

Default constructor: `imgData_construct_dataset`

Field	Description	Default value	Type
camera	The camera which datasets was taken from.	See 5.8.1.	`Camera` struct.
ipOutput	A list of image processing output parameters for external applications. Note: ImgMan does not read it.	[] (empty)	Row cell array of `ipOuput` structs (see 5.8.6).

ipParam	A set of image processing parameters for external applications. Note: ImgMan does not read this field.	[] (empty)	`IpParam` struct (see 5.8.7).
isValid	A flag indicating the validity of the dataset. ImgBrowser does not show invalid datasets.	1	Boolean.
label	The label of this dataset.	'n/a'	String.
masterCropArea	The area of the master crop region.	[]	Four-element row array of doubles, [xmin ymin width height] each in default spatial coordinates.
nrBgImgs	Number of background images in the dataset. ImgMan treats the first `nrBgImgs` images in the dataset as background images.	0	Integer.
nrBeamImgs	Number of beam images in the dataset.	0	Integer.
rawImg	A list of raw image parameters.	[] (empty)	Row cell array of `rawImg` structs (see 5.8.8).

6.8.3 ImgAnalysisData

Default constructor: `imgData_construct_imgAnalysis`

Field	Description	Default value	Type
dsIndex	The index of a dataset (which doesn't have to be valid).	1	Integer.
imgIndex	The index of an image in the selected dataset.	1	Integer.

ipOutput	A set of image processing output parameters.	[] (empty)	`IpOutput` struct (see 5.8.6).
ipParam	A set of image processing parameters.	Default structure.	`IpParam` struct (see 5.8.7)

6.8.4 ImgBrowserData

Default constructor: `imgData_construct_imgBrowser`

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

6.8.5 ImgManData

Default constructor: `imgData_construct_imgMan`

Field	Description	Default value	Type
dataset	A list of raw image datasets.	[] (empty)	Row cell array of `dataset` structs (see 5.8.2).
ipOutputChanged	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.	0	Boolean.
isDirty	A flag indicating whether the datasets need to be saved.	0	Boolean.

6.8.6 IpOutput

Default constructor: `imgData_construct_ipOutput`

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

6.8.7 IpParam

Default constructor: `imgData_construct_ipParam`

Field	Description	Default value	Type

algIndex	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.	1	Integer.
annotation .centroid .current.color	Color of the current beam centroid mark.	[0 0 0] (black)	RGB color.
annotation .centroid .current.flag	Flag indicating whether to display the current beam centroid mark.	0	Boolean.
annotation .centroid .goldenOrbit.color	Color of the golden orbit beam centroid mark.	[1 0.84 0] (gold)	RGB color.
annotation .centroid .goldenOrbit.flag	Flag indicating whether to display the golden orbit beam centroid mark.	0	Boolean.
annotation .centroid .laserBeam.color	Color of the laser beam centroid mark.	[1 0 0] (red)	RGB color.
annotation .centroid .laserBeam.flag	Flag indicating whether to display the laser beam centorid mark.	0	Boolean.
beamSizeScaleUnit	The scale unit of the beam size.	'pix'	'pix', or 'mm'
colormapFcn	Name of the function that creates a colormap.	'jet'	String
crop .auto .custom	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.	Auto = 0 Custom = 0	Boolean.

Filter .floor .median	Flags indicating whether a filter should be applied to the raw image.	Floor = 0 Median = 0	Boolean.
lineWidthFactor	A factor for the width of the annotation lines (in proportion to the size of the corresponding axes).	1/150	Double.
slice .index .plane .total	The current index, plane, and total number of slices of the raw image. ImgMan extracts beam data from the specified slice only.	Index = 1 Plane = 'x' Total = 1	Index: Integer. Plane: 'x' or 'y'. Total: Integer.
nrColors .max .min .val	Maximum, minimum, and the current number of colors in the colormap.	Max = 4084; Min = 64; Val = 256;	Integer.
subtractBg .acquired .calculated	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.	Acquired = 0 Calculated = 0	Boolean.

6.8.8 RawImg

Default constructor: `imgData_construct_rawImg`

Field	Description	Default value	Type

customCropArea	Coordinates of a custom region of interest for the raw image.	[] (empty)	Four-element row array of doubles, [xmin ymin width height] each in default spatial coordinates.
data	Grayscale image data that ImgMan retrieves from an IOC.	0	Matrix of integers.
ignore	A flag for ignoring the raw image, overriding the validity field of an instance of `IpOutput` (see 6.8.6).	[]	Boolean or [].
timestamp	The time of when the camera delivered the raw image to the IOC.	-1	LabCA timestamp.

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

`lca2matlabTime(lcaTS)`

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

Parameter
lcaTS
  LabCA timestamp as acquired through `lcaGet`.

Example

```
>> [value, lcaTS] = lcaGet('MIKE:BEAM');
>> matlabTS = lca2matlabTime(lcaTS);
>> datestr(matlabTS, 'mmm dd HH:MM:SS.FFF')

ans =

Aug 23 18:04:18.658
```

8.2 lcaTs2PulseId

`lcaTs2PulseId(lcaTS)`

Extracts pulse ID from labCA timestamp
Parameter
lcaTS
  LabCA timestamp as acquired through `lcaGet`.

Example

```
>> [value, lcaTS] = lcaGet('MIKE:BEAM');
>> lcaTs2PulseId(lcaTS);
ans =

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');
```

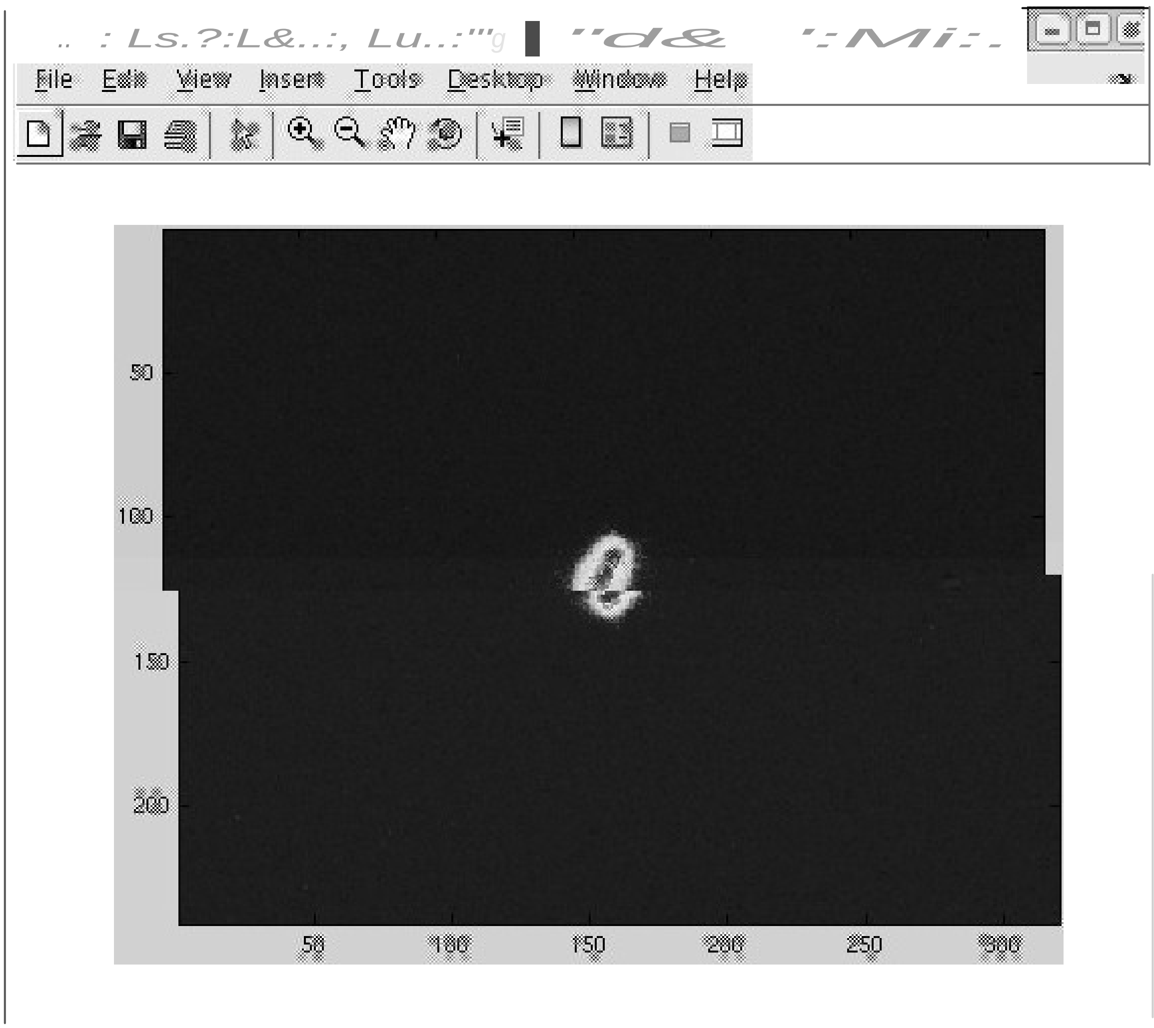
Reshape into a 2D image

```
>> img = reshape(img, 240, 320);
```

Display image

>> image (img)

Result



Appendix A EPICS/SLC attributes

9.1 Magnet attributes

(Note: some might not be available yet)

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

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