# Camera Visualization Concept of Operations

December 10, 2016: Third draft

The purpose of this document is to describe the way that camera visualization will operate. By comparing this document with a similar one from Data Management (DM) we hope to identify common issues and solutions – one goal being avoiding duplication of effort. For clarity, the relationships are divided into three categories:

• User interface

• Functionality

• Relation to other LSST software (*e.g.*, camera data acquisition and the LSST stack).

### Visualization architecture

The camera visualization consists of four major components, which can run on two platforms, back end (BE) and front end (FE), although separation is not required. The BE has access to the camera images and other data repositories, possibly via a data butler, and implements functions that require direct access to the raw pixel values. The FE implements the user interface – command input and image displays and other output. It does not directly access any data repositories.

• Data butler. This SLAC-written code has access to the image repositories. It retrieves requested images and reformats them (if necessary) into single-extension FITS files.  
Camera-specific issues: For real-time needs, a short latency is important. Firefly requires a single-extension FITS file, but this may go away.

• Firefly. This IPAC-written code provides the primary image display functionality and some user manipulations. It has two parts, one on the BE and one on the FE. Its BE part turns the FITS file into an image that is sent to and displayed by the FE component, which may reside on a remote computer (*e.g.*, over the internet). The FE displays images and also implements some user interaction, described below.

• FE user interface. UIUC-written JavaScript provides command entry, window management, and help. To assist non-expert users, command entry is assisted by dynamic templates. Commands can be entered from a command line or from a customized Firefly toolbar. Firefly provides an API that enables JavaScript enhancements to its image display and data access functionality.  
Camera-specific issue: The user interface is not designed for general-purpose image analysis. We have not looked into its applicability to that use. For example, we have not attempted to implement a command syntax that will implement complex data analysis functions.

• BE analysis code. Written by UIUC, it implements algorithms that require access to the raw image pixels, and other functionality that may require access to data repositories. To the extent possible, these functions make use of the LSST stack.  
Camera-specific issue: Performance specifications will vary with use case.

### Front end user interface

The user interacts via web browser, with functionality implemented in JavaScript. The URL points to the back end server, which must already be running. Low bandwidth communication between the FE and BE allow the user and the display to be remote from the database and data processing. Commands that do not require processing of raw image data or access to other remote data (*e.g.*, region selection on a displayed image) are performed locally, either by Firefly or the UIUC JavaScript. The user can create windows as needed, so that, for example, multiple images or analysis results can be simultaneously displayed. Help is available interactively and during command entry.

### Functionality

The set of implemented functions are determined by camera needs, so they will be described here in some detail. That will enable the identification of needs that are common to the various visualization environments.

#### Functions that are needed in real-time (as data is taken):

• Display latest:

Display the most recent image that has been taken. This involves coordinated operation by the FE, BE, and Firefly. The BE will inform the FE whenever a new image is received. When “display latest” is active the FE will, after a user-specified latency, tell Firefly to display the newest image. Latency gives the user time to respond, if desired. In particular, the user can pause updating in order to perform more detailed studies of an interesting image.

#### Functions that display or analyze a single image:

• Display (a specified image):  
The user can specify the image via URL or by selection criteria that the data butler will interpret appropriately. This latter functionality is not yet implemented. The user can specify whether or not the overscan pixels will be displayed.

• Select a region of an image:  
Many functions operate on a specified region of an image. Firefly supports some interactive region selection (points, lines, and rectangles, with circles to come). The user interface has added the selection of hardware regions (overscan and/or data, as well as amplifiers and CCDs). On the command line, a region can be specified by pixel coordinates or by region name (*e.g.,* the name of a particular amplifier). When necessary, the selected region is passed to the BE as a command parameter that specifies the pixels to be processed.  
Camera-specific issue: We will want to be able to specify the cursor’s hardware region name as an analysis command parameter.

• Display the pixel coordinates, and the hardware region (amplifier and CCD) as the user moves the cursor.  
Camera-specific issue: When zoomed out, FF does not merge pixels, it displays a random pixel. This degrades HW diagnostic capability.

• Overlay the hardware (amplifier and CCD) boundaries on an image.

#### Functions that display or analyze multiple images:

• Display (another specified image)  
After creating another window, the user can request that another image be displayed in it.  
TBD: The second image can be specified as having a relationship (*e.g.*, DAQ conditions) to the previously displayed image.

• Blink two images.  
Firefly supports the ability to flip the display between multiple images.

• Display the difference or ratio of two images.  
This is implemented in the BE python code. A new image is created, and displayed. Note that the Firefly architecture requires that image display commands be initiated in the FE, so the URL of the new image must be passed back to the FE.

### Relation to other LSST software

For reliability and performance, we plan to use as much of the LSST stack functionality as possible. The back end environment implements the appropriate interfaces, but, because we have not yet written many complex analysis functions, we have not yet used the available functionality.

**Summary of** [**Paul O’Connor’s Visualization Feature Wish List**](https://community.lsst.org/t/paul-oconnors-visualization-feature-wish-list/545) (with annotations)

• Continuous coordinate and ADU readout under mouse pointer  
Firefly can do this and pass the result to JS for display.  
UIUC has implemented a “display\_mouse” command..  
When zoomed out, FF does not merge pixels, it displays a random pixel

• Mosaic display of multi-extension FITS files with/without overscan displayed  
SLAC code will do this. FF requires single extension FITS format for display.

• Magnifier and full image thumbnail windows  
FF does this.

• Scroll-wheel zoom  
FF does not do it, but could, with the caveat that zooming is not continuous; zoom levels are predefined.

• Standard, selectable zoom levels  
FF does this

• Middle-mouse pan  
FF does panning by dragging

• Rotate and reflect   
FF does this with Rotate and Flip commands

• Display pixel table  
FF does not do this. It gives dynamic readout.

• Scale contrast to max/min of pixels in region  
UIUC code will do this.

• Contrast scale segment-by-segment  
UIUC (or SLAC) code will do this.

• Zscale mapping   
FF supports zscale stretch

• Asinh, sinh, log scaling  
FF supports log stretch

• Import regions files with circle/box display  
FF supports ds9 regions

• Multi-frame functionality: coordinate and scale locking, blinking,  
FF supports multiple images in Expanded View. You can WCS align and blink them in single image view with Auto Play.

• User-controlled tiling  
What control is wanted?

• Annotation  
FF supports markers (circle with label), you can save them. More work is being done on it for other projects.

• Colormap options  
FF supports all ds9 color maps and legacy Spitzer maps. They will be revamped as none of them are good enough.

• vi-like keystroke commands tied to mouse pointing (see e.g. https://www.noao.edu/kpno/manuals/ice/node31.html)  
FF does not support it.  
UIUC has implemented this.

• Pop-up graphics windows  
FF Expanded View is a pop-up.

• Select region, display statistics, display histogram with mean and sd  
FF does allow selecting region and displaying statistics, histogram can be added later.  
UIUC has begun to implement histograms.

• Aperture photometry -- define aperture, point at source, get background-subtracted counts, get radial profile with fit, fit Gaussian. Averaged row/column traces with limits  
This is a UIUC TBD.

• 3D surface/lego, contour map display

• Image math (sum, difference, quotient of 2 or more images) and display of result  
We need to make a list of desired window/display types, and desired parameters and user interactions.  
Being implemented at UIUC.

• Configuration file for defaults  
FITS visualization parameters define the default configuration.  
UIUC is implementing a configuration file.

• Standalone operation  
What does this mean?

• Support for linux and windows  
FF should be working in IE, Chrome, and Firefox on Windows.   
This has been verified on MacOS also.