CASS – CFEL-ASG Software Suite

Lutz Foucar[a,b,*], Anton Barty[c], Nicola Coppola[c,d], Robert Hartmann[e], Peter Holl[e], Uwe Hoppe[f], Stephan Kassemeyer[b], Nils Kimmel[g,h], Jochen Küpper[f,a,c,i], Mirko Scholz[j], Simone Techert[a,j], Thomas A. White[c], Lothar Strüder[a,g,h,k], Joachim Ullrich[a,l,m]

[a] *Max Planck Advanced Study Group, Center for Free Electron Laser Science (CFEL), Notkestraße 85, 22607 Hamburg, Germany*
[b] *Max-Planck-Institut für medizinische Forschung, Jahnstraße 29, 69120 Heidelberg, Germany*
[c] *Center for Free-Electron Laser Science (CFEL), Deutsches Elektronen-Synchrotron DESY, Notkestraße 85, 22607 Hamburg, Germany*
[d] *European XFEL GmbH, Albert-Einstein-Ring 19, 22761 Hamburg, Germany*
[e] *PNSensor GmbH, Römerstraße 28, 80803 München, Germany*
[f] *Fritz-Haber-Institut der MPG, Faradayweg 4-6, 14195 Berlin, Germany*
[g] *MPI Halbleiterlabor, Otto-Hahn-Ring 6, 81739 München, Germany*
[h] *Max-Planck-Institut für extraterrestrische Physik, Giessenbachstraße, 85741 Garching, Germany*
[i] *University of Hamburg, Department of Physics, Luruper Chaussee 149, 22761 Hamburg, Germany*
[j] *Max-Planck-Institute for Biophysical Chemistry, Am Faßberg 11, 37077 Göttingen, Germany*
[k] *University of Siegen, Emmy-Noether Campus, Walter Flex Straße 3, 57068 Siegen, Germany*
[l] *Physikalisch-Technische Bundesanstalt, Bundesallee 100, 38116 Braunschweig, Germany*
[m] *Max-Planck-Institut für Kernphysik, Saupfercheckweg 1, 69117 Heidelberg, Germany*

Abstract

The Max Planck Advanced Study Group (ASG) at the Center for Free Electron Laser Science (CFEL) has created the CFEL ASG Software Suite *CASS* to view, process and analyse multi-parameter experimental data acquired at Free Electron Lasers (FELs) using the CFEL ASG Multi Purpose (CAMP) instrument [1].
The software is based on a modular design so that it can be adjusted to

*Corresponding author
Email address: lutz.foucar@asg.mpg.de (Lutz Foucar)

accommodate the needs of all the various experiments that are conducted with the CAMP instrument. In fact, this allows the use of the software in all experiments where multiple detectors are involved. One of the key aspects of *CASS* is that it can be used either 'on-line', using a live data stream from the free-electron laser facility's data acquisition system to guide the experiment, and 'off-line', on data acquired from a previous experiment which has been saved to file.

Keywords: C++, Object-oriented, Data analysis, multithreaded, modular

---

PROGRAM SUMMARY

Manuscript Title:CASS – CFEL-ASG Software Suite
Authors:Lutz Foucar, Anton Barty, Nicola Coppola, Robert Hartmann, Peter Holl, Uwe Hoppe, Stephan Kassemeyer, Nils Kimmel, Jochen Küpper, Mirko Scholz, Simone Techert, Thomas A. White, Lothar Strüder, Joachim Ullrich
Program Title: CASS
Journal Reference:
Catalogue identifier:
Licensing provisions:GNU General Public License, version 3
Programming language:C++
Computer:Intel x86-64
Operating system:GNU/Linux (for information about restrictions see outlook)
RAM: $>$ 8 Gbytes
Number of processors used: user choosable, by default runs with $>$ 16 threads
Supplementary material:
Keywords: Object-oriented, Data analysis, multithreadded, modular, Free-electron-lasers
Classification: 2.3 Experimental Analysis, 3 Biology and Molecular Biology, 15 Laser Physics, 16.4 Experimental Analysis,
External routines/libraries: Qt-Framework[1], SOAP[2], (optional HDF5[3], VIGRA[4], ROOT[5], QWT[6])
Subprograms used:
Nature of problem:
Analysis and visualization of scientific data acquired at Free-Electron-Lasers
Solution method:
Generalize data access and storage so that a variety of small programming pieces can be linked to form a complex analysis chain.
Restrictions:

Unusual features:

complex analysis chains can be build without recompiling program
Additional comments:
An updated extensive documentation of CASS is available at [7]
Running time:
Depending on the data size and complexity of analysis algorithms.

[1] http://qt.nokia.com

[2] http://www.cs.fsu.edu/~engelen/soap.html

[3] http://www.hdfgroup.org/HDF5/

[4] http://hci.iwr.uni-heidelberg.de/vigra/

[5] http://root.cern.ch

[6] http://qwt.sourceforge.net/

[7] http://www.mpi-hd.mpg.de/personalhomes/gitasg/cass

1. Introduction

The fourth generation Free Electron Laser (FEL) light sources deliver ultra-short, coherent and intense pulses of radiation extending from the VUV to the X-ray regime. This allows the investigation of a plethora of new phenomena in physics, biology and chemistry that were not accessible using synchrotron radiation sources. In order to cover a large part of the huge variety of different experiments possible with these new light sources the Max Planck Advanced Study Group (ASG) at the Center for Free Electron Laser Science (CFEL) designed the CFEL ASG Multi Purpose (CAMP) instrument, a multi-purpose apparatus that is fashioned to implement a large number of different sample delivery and detection devices in essentially any combination. With this, experiments as different as the determination of structural data of nanocrystals[2] and the investigation of reaction dynamics of molecules[3] can be performed in the same instrument. In order to monitor the results, analyse the experiments and process the multi-parameter correlated data taken by the various detectors the need for a new software package arose.

Since a large data volume is generated by the detectors for each FEL pulse, the software must achieve a high processing rate and, as a result, has to be capable of making use of multi-core computers. Moreover, the user demands of the multi-purpose instrument vary significantly such that the program should be modular and independent of the underlying operating system. To accommodate these requirements, C++ was chosen as an appropriate development language using the Qt Framework [4].

In order to provide a wide range of analytical possibilities and to allow the users to alter the flow of the analysis according to information which might only become available during the course of individual experiments, a modular architecture was used in which small processing units can be linked to form the overall processing pipeline. The choice of processing units and the connections between them can be changed 'on the fly', either when operating 'on-line' using a live data stream from the free-electron laser facility's data acquisition (DAQ) system, or 'off-line' processing the data from a completed experiment.

The software has been named *CASS*, the CFEL-ASG Software Suite. It is developed within the CFEL collaboration under the leadership of the ASG under the GPL v3 licence.

2. Description

CASS consists of two basic modules which are coupled by a ring buffer (described in detail in Section 2.3): the input and the processing module described in Section 2.2 and 2.5 respectively. A schematic description is shown in figure 1. This separation disentangles the data input from the analysis and therefore enables the program to even process large scale data that are produced in bursts for each light pulse.

The currently available input modules receive the data, either from the FEL facility's DAQ system or from a file, and store the information in the *CASS* event class format, the *CASSEvent*, described in the next section. The processing module then evaluates the data within the *CASSEvent* and stores the results of the analysis in special data containers, which are described in detail in Section 2.4. To ensure that this task is done as quickly as possible, *CASS* must work on multiple *CASSEvents* simultaneously and therefore holds a list of processing modules, which can run in parallel.

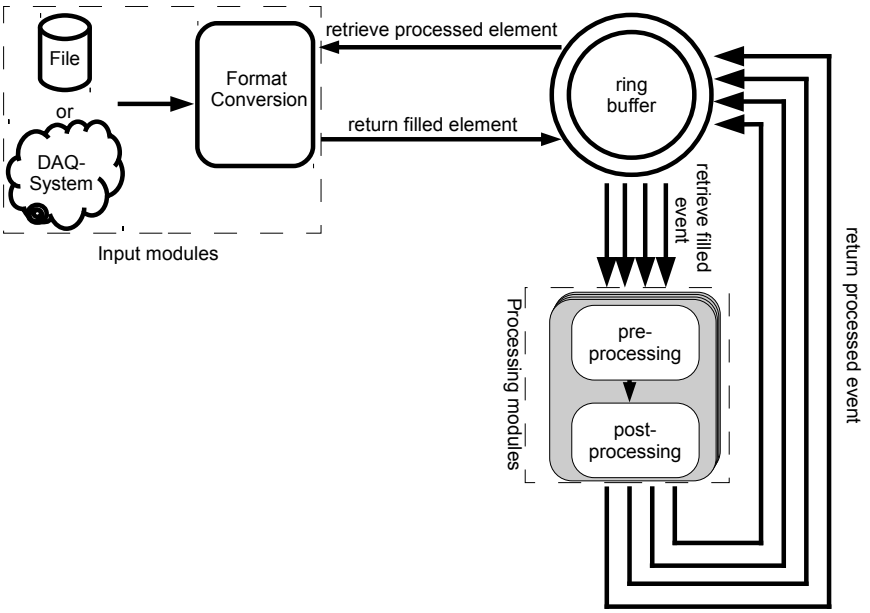

Figure 1: Overview of the general layout of *CASS*. The arrows mark the flow of the *CASSEvents*. Details are described in the text.

2.1. The CASSEvent

It is necessary to bundle the information from all detectors for each X-ray laser pulse individually to be able to investigate and correlate the recorded data. For this reason a special container, named *CASSEvent*, was created that ties all the available information together, marked with a unique identification number to distinguish it unambiguously from other events.

Since the container has to hold a variety of different data types and structures, the informations of the individual detectors are kept in device classes specific to the distinct detector. The *CASSEvent* itself encloses a list of these device classes.

As of now there are five different data sources that deliver data from the different detectors used in CAMP, machine-related data, Acqiris analogue-to-digital-converters (ADC)[5], Acqiris time-to-digital-converters (TDC)[6], commercially available charge coupled device (CCD) and pnCCD cameras. pnCCDs are special large-area, fast read-out p-n junction CCD X-ray detectors that have been developed for FEL applications and are an essential part of the CAMP instrument [1]. Currently two types of commercially available CCDs are supported, the Opal CCD[7] and Pulnix CCD[8]. The dedicated

device classes to store the information from these different sources named 'MachineDevice', 'Acqiris', 'Acqiris', 'pnCCD' and 'CCD' are described below.

pnCCD and CCD The data-structure of the pnCCD and the commercially available CCDs do not differ, therefore the data of both are stored in a general 'pixel-detector' class. The separation into two different device classes is historically given and obsolete and will therefore be merged into one device in future releases of *CASS*.

To accommodate a set of detectors used in a dedicated experiment the 'pnCCD' and 'CCD' device classes contain an array of 'pixel-detector' objects, which enclose the frame data of one camera. An individual frame is a matrix of values, representing the pixel array. In addition to the pixel data itself, information on the parameters of the specific frame such as the dimensions of the detector are included.

Since the frame data is usually large it will be time consuming to analyse. To speed up consecutive analysis steps, the frame data can be reduced to only the interesting information for an individual experiment. This concludes that there is a need for a compressed representation of the frame and therefore the 'pixel-detector' class embodies a list of 'pixel' classes, which contain the coordinates of an individual pixel within the frame and its value.

Acqiris ADC and TDC The *CAMP* instrument can incorporate various types of time-of-flight and delay-line detectors. Here data reading proceeds via Acqiris ADC and TDC instruments.

Because multiple ADC and TDC instruments might be present in an individual experiment, the Acqiris TDC and Acqiris ADC device classes contain lists of instrument classes. Since each instrument has multiple channels to record the information from the detectors each instrument class consists of a collection of channel classes in which the recorded data is stored.

Machine-related Data The machine data class describes information that is available about the light source and the endstation itself, such as the intensity of the individual X-ray pulse or the setting of the monochromator. Many of these parameters vary on a shot-by-shot basis, such that a preset or an average value for all events would not be appropriate.

2.2. Input

The role of the input module is to collect raw data from a source, convert it and fill the next available *CASSEvent* in the ring buffer (see Section 2.3).

Currently, there are two input modules for the use at the X-ray laser facility Linac Coherent Light Source (LCLS)[9] available in *CASS*, one taking data from a file on disk and the other from LCLS's DAQ system, which works through a POSIX shared memory area. Additional input modules are in development to adapt to different file formats or DAQ systems at other facilities, e.g. FLASH [10], the European XFEL [11] or SACLA [12].

At LCLS, the data format used for both, the shared memory and file input paths, is called extended tagged container (XTC)[13]. It contains the bundled information that was recorded by the DAQ system for every X-ray laser shot. A set of format converters within the *CASS* input modules transform the data from the XTC format to the *CASSEvent* format. The user can selectively activate only those converters that are related to data which is relevant to the distinct experiment. This is important since not all of the information available will be necessary for any individual analysis task, and avoiding the conversion of irrelevant data improves the performance of the program.

2.3. The Ring Buffer

As described above the input and the analysis module of *CASS* have been separated from each other to enable *CASS* to process bursts of events. This implies that the input module needs to pass the data to the processing module through a buffer, ideally a ring buffer.

The elements of the ring buffer used within *CASS* are *CASSEvents*. The buffer can operate in either a blocking or a non blocking mode. In blocking mode it ensures that elements are processed before they are refilled with new data, meaning that all data from the input component is guaranteed to be analysed. This situation is desirable for off-line data evaluation, but would be a disadvantage for on-line data processing where the time taken for analysis could easily be longer than the time available before the arrival of the next event. The non-blocking mode is provided for this situation, and operates by allowing ring buffer elements which have been filled but not yet been evaluated to be reused as soon as newer event data becomes available.

Since the input and the multiple analysis modules run in separate threads it is possible that the same element is addressed by different components of the program simultaneously. To prevent this, a fine grained access to the

CASSEvents is needed. Therefore, the individual elements of the buffer can only be addressed through a special access interface, which uses flags that are inherent to the ring buffer elements to distinguish their state.

In blocking mode, the ring buffer ensures that no new elements can be retrieved before all previous events have been processed. In non-blocking mode, it is allowed to reuse elements which have been filled but not yet been processed. When the ring buffer operates in this way, the oldest events are always chosen to be overwritten first.

2.4. Results Data Container

CASS has a special data container to store the results of the analysis, given the general name *result-container*. *Result-containers* in *CASS* can have zero, one or two dimensions.

Result-containers of zero dimensionality correspond to individual scalar values to store, for example, a single reading from a photo-diode. In addition, scalar values can be used as boolean values to control the behaviour of other *post-processors*, for example by inhibiting their action.

One dimensional *result-containers* are for storing data such as traces from digitizers, the history of scalar values or histograms. To be able to see how many of the values did not fall within the defined range of the one dimensional *result-container*, it contains information about under- and overflow statistics.

Two dimensional *result-containers* might be applied for storing images and 2d histograms, where under- and overflow statistics are kept.

2.5. Analysis

The analysis module runs with multiple threads so that *CASSEvents* can be processed in parallel, and the number of threads can be defined by the user to fully exploit the available resources. Each thread will work on only one *CASSEvent* and can exclusively access and process the data contained in this ring buffer element.

The analysis begins with the retrieval of the next processable *CASSEvent* from the ring buffer. It is then passed through the pre-processing framework which is described in detail in the next section. The pre-processed *CASS-Event* is transferred to the post-processing framework described in section 2.5.2. After the completion of both tasks, the ring buffer is notified that the *CASSEvent* is fully processed so that it can be reused.

2.5.1. Pre-Processing

The pre-processing framework should refine the data such that it can be easily evaluated in following analysis steps. It is defined by special pre-processing units called *pre-processors*. To optimize the analysis speed the user can selectively choose which of the *pre-processors* should treat the *CASSEvent*. Presently there are two pre-processing elements available, both for handling the data from the different CCD detectors used in the CAMP instrument, pnCCD and commercially available CCD.

The *pre-processor* which operates on the pnCCD camera data performs all the required corrections to render the raw pixel data quantitative so that individual photons can be detected and spectral information extracted[14]. The *pre-processor*, that processes the data from the commercially available CCD cameras thresholds the pixels to reduce the amount of pixels evaluated by subsequent analysis steps.

The pre-processing tasks could equally be performed by the more modular post-processing framework, described below, with no loss of functionality, but have historically been implemented in the pre-processing framework. In a future version of *CASS*, those tasks will be merged into the post-processing framework.

2.5.2. Post-Processing

The post-processing framework consists of individual elements, called *post-processors*, which pass data among themselves using the *result-container* data structure. The framework allows the list of active processing elements and the flow of data between them to be defined 'on the fly' without the requirement to recompile or even to re-start the program. Figure 2 shows two examples of such user defined program flows.

An individual analysis chain is defined by a plain text configuration file that follows the well-known '.ini file' format. All possible dependencies among the *post-processors* are resolved automatically when the program flow is set up.

Post-processors can retrieve the data from the *CASSEvent* and put it into a *result-container*. Once the data has been stored in a *result-container*, further *post-processors* can access the data without having to be aware of the details of the *CASSEvent* structure. This allows for a high degree of generality and code reuse in the analysis framework.

Any *post-processor* performs an individual task, ranging from basic logical and arithmetic, used to combine results or control flow, to advanced

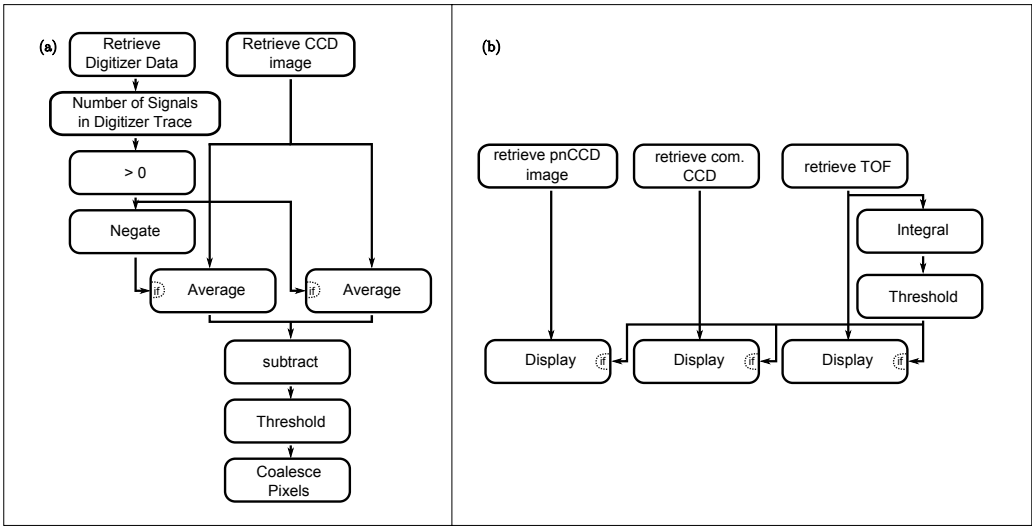

Figure 2: Two examples of *post-processor* analysis chains. (a) To identify individual photons on a CCD detector during an experiment involving very low scattering intensity. (b) To view the correlated information from the different detectors for selected events during the experiment.

analytical tasks. All of them can be set up to perform the task conditionally, where the decision whether to analyse the event is based on the boolean result of another *post-processor*. Individual *post-processor* can be multiply instantiated and, hence, re-used with different parameters or inputs.

More complicated analysis pipelines may require that the result of one *post-processor* is fed to more than one downstream *post-processor*. To avoid performing the same analysis more than once on any piece of data, a memoisation technique is used in which each *post-processor* caches the result of the evaluated *CASSEvent*.

2.6. Viewing the data

It is advantageous to disentangle the data display from the data evaluation, therefore *CASS* itself is a pure command-line application and is unable to display the results of the post-processing or the recorded data itself. To view the evaluated data, each result of a *post-processor* can be streamed over TCP/IP to displaying clients.

In order to transfer the hierarchical data contained in *result-container* objects, they have to be serialised. *CASS* provides *serialiser*/*deserialiser* classes that can be used to add serialisation capabilities to any object. After

serialisation, *CASS* sends the data to the requesting client application using the Simple Object Access Protocol (SOAP). The implementation in *CASS* uses the *gSOAP* [15] [16] library.

This TCP/IP streaming framework can be used by Graphical User Interface (GUI) clients to obtain results for display. Besides simply viewing the data, the client can be used to control the execution of *CASS* by sending commands, for instance instructing it to re-read its configuration file and alter the analysis pipeline.

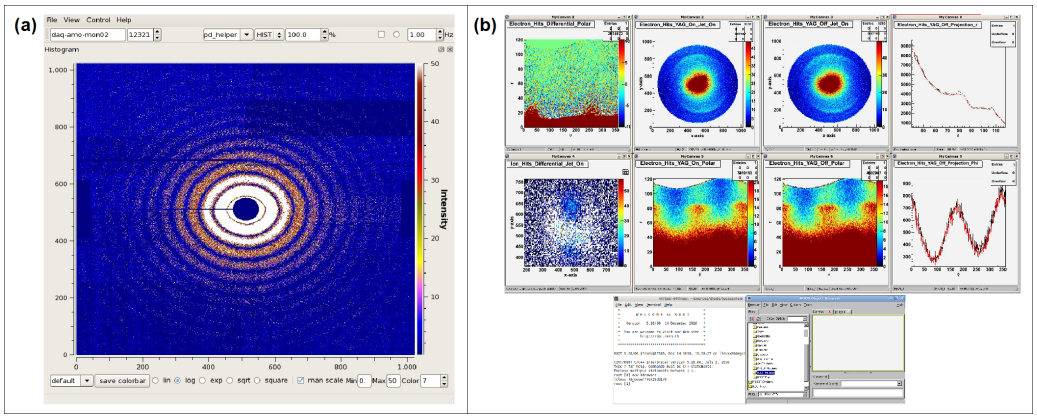

Figure 3: (a) Screen shot of *joCASSView*, displaying an diffracted image of a water droplet on the pnCCD. (b) Screen shot of *luCASSView* showing the results and intermediate of a complex analysis pipline starting with data from a commercial CCD.

So far there are two GUI applications available, described in the next two sections. Figure 3 shows screen-shots of both viewers. Recently a third option has been added where a webserver can be compiled into *CASS* that will serve the data as a web page.

2.7. joCASSView

joCASSView is a GUI that displays the data contained in *result-container* objects as plots or images. Since *joCASSView* requests the data over TCP/IP as described in Section 2.6, the viewer can be run from a different machine than the *CASS* server and multiple viewer instances might be opened to view the data at the same time. The *result-containers* in the current analysis pipeline are shown in a drop-down menu and one can be selected for display at a time. To simplify the menu, *result-containers* which are only required as intermediate steps in the *post-processor* chain can be marked as 'hidden'

in *CASS*, in which case they will not be shown in the GUI. *joCASSView* is written in a platform-independent code using the Qt[4] framework, where graph plotting is done with help of the Qwt[17] library, which allows to display images with a high degree of responsiveness. Currently *joCASSView* can only display data. Performing operations such as projections on the viewed data via the GUI is not possible.

2.8. luCASSView

To overcome the limited possibilities of *joCASSView*, a second viewer, named *luCASSView*, was created, which is based on ROOT [18]. ROOT allows the user to interact with the viewed data via its integrated C++ interpreter and macro capability.

luCASSView retrieves the *result-containers* from *CASS* via the TCP/IP interface described above, displays and optionally saves them into a ROOT file. To view the data contained in the *result-containers*, its contents are copied into ROOT histograms, which are then displayed using the ROOT framework. Data is only requested and transferred via TCP/IP for updating only when the user is displaying the corresponding ROOT histogram, limiting the bandwidth required for the data transport.

3. Installation Instructions

3.1. Getting the sources

The *CASS* project makes use of the version control program Git [19]. A tarball is automatically created from the latest version of the master branch every three hours. One can download it from the location `http://www.mpi-hd.mpg.de/personalhomes/gitasg/Downloads/cass.latest.tar.gz`. Older, stable versions are available as tags in the Git repository. Their tarball download location can be found in the documentation [20].

3.2. Compiling

CASS makes use of the Qt[4] makefile generation tool, qmake. A separate settings file, that is read by the qmake project file, allows the user to enable or disable modules of *CASS* or switch between different modes, e.g. creating 'on-line' or 'off-line' binaries of *CASS*.

Compiling and installing *CASS* is performed by a standard sequence of commands:*qmake;make;make install;*

The libraries and tools required for building *CASS* are Qt[4] version 4.6.2 or later, which is used for the signal/slot concept and configuration file parsing. For streaming objects over TCP/IP, gSOAP [15] [16] version 2.7.17 or later is needed

Some of the optional features require additional libraries:

- Qwt[17] library version 5.2.0 or later (but no later than 6.0.0), which is used for viewing the data via the *joCASSView* GUI client.
- HDF5[21] library version 1.8.0 or later, which is used to save computed results in files in the Hierarchical Data Format (HDF5).
- ROOT[18] version 5.28.00c or later, which is used to save computed results and for viewing the data via *luCASSView*.
- VIGRA[22] library version 1.7.0 or later, which can be used for advanced image analysis.

4. Documentation

The documentation for *CASS* is generated automatically from the most recent stable source code every three hours using the source code parsing tool doxygen [23] and can be found at `http://www.mpi-hd.mpg.de/personalhomes/gitasg/cass`[20]. Besides a complete description of *CASS*, an overview of all available *post-processor* types is given, along with all the possible parameters for each *post-processor*. Some example configuration files are provided to guide the user or to act as a basis for the development of more complicated analysis pipelines.

5. Alternative Software

Alternative software suites are available for analysing data acquired using the LCLS, named psana [24] and pyana [25]. The former is an object-oriented C++ framework, whereas the latter is a Python interface.

In addition, users of LCLS have developed a software package, named 'Cheetah' [26], which performs screening of nanocrystal data from the CXI endstation.

The authors are not aware of an existing software framework that has capabilities similar to *CASS* at the FLASH experimental site.

6. Outlook

An important missing feature is the event-awareness of viewer applications. While the analysis pipeline of *CASS* is able to relate different measurements and computations to one another within one *FEL* event, the viewers can only request the most recent results which do not necessarily match with regard to their event id. This causes problems if, for example, an experiment relies on monitoring relationships between the data of different detectors in a single FEL pulse.

CASS is currently tightly coupled to the LCLS pdsdata libraries[13], which are needed to read the LCLS XTC data format and to connect to the shared memory area in 'on-line' mode. Unfortunately, these libraries can only be compiled under Unix derivatives with POSIX shared memory, e.g. Linux[27]. To enable the use of *CASS* for all available operating system platforms, it will be necessary to decouple the LCLS libraries from the core program.

7. Contributions

L.F., J.K. and N.C. created the fundamental design. L.F., N.C., U.H., S.K., N.K., J.K., M.S. and T.A.W wrote code. L.F., R.H., P.H. and J.K. gave advice on implementation details. L.F., A.B., J.K., S.T., L.S. and J.U. gave advice on usage patterns/requirements.

Acknowledgements

The support and patience of Chris O'Grady and Matt Weaver in describing the layout of the XTC data format is acknowledged. We thank Amedeo Perazzo and his team for their support in running the software on the LCLS hardware. The authors are grateful to Ilme Schlichting for comments on the manuscript. Per Johnsson, Arnaud Rouzee and Marc Vrakking contributed code for the analysis of commercial CCD camera images. Marcus Adolph, Christoph Bostedt, Tomas Ekeberg, Benjamin Erk, Tais Gorkhover, Andrew Martin, Daniel Rolles, Benedikt Rudek and Stephan Stern tested *CASS*, gave important feedback and suggested improvements. We acknowledge support from the Max Planck Society for funding the development of *CASS*. M.S. is grateful to SFB 755 for financial support.

[1] L. Strüder, S. Epp, D. Rolles, R. Hartmann, P. Holl, G. Lutz, H. Soltau, R. Eckart, C. Reich, K. Heinzinger, C. Thamm, A. Rudenko, F. Krasniqi, K.-U. Kühnel, C. Bauer, C.-D. Schröter, R. Moshammer, S. Techert, D. Miessner, M. Porro, O. Hälker, N. Meidinger, N. Kimmel, R. Andritschke, F. Schopper, G. Weidenspointner, A. Ziegler, D. Pietschner, S. Herrmann, U. Pietsch, A. Walenta, W. Leitenberger, C. Bostedt, T. Möller, D. Rupp, M. Adolph, H. Graafsma, H. Hirsemann, K. Gärtner, R. Richter, L. Foucar, R. L. Shoeman, I. Schlichting, J. Ullrich, Large-format, high-speed, x-ray pnccds combined with electron and ion imaging spectrometers in a multipurpose chamber for experiments at 4th generation light sources, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 614 (3) (2010) 483 – 496. doi:DOI: 10.1016/j.nima.2009.12.053.
URL http://www.sciencedirect.com/science/article/B6TJM-4Y35TDM-2/2/a419ba61e849f00a1fc9131459669ae0

[2] H. N. Chapman, P. Fromme, A. Barty, T. A. White, R. A. Kirian, A. Aquila, M. S. Hunter, J. Schulz, D. P. DePonte, U. Weierstall, R. B. Doak, F. R. N. C. Maia, A. V. Martin, I. Schlichting, L. Lomb, N. Coppola, R. L. Shoeman, S. W. Epp, R. Hartmann, D. Rolles, A. Rudenko, L. Foucar, N. Kimmel, G. Weidenspointner, P. Holl, M. Liang, M. Barthelmess, C. Caleman, S. Boutet, M. J. Bogan, J. Krzywinski, C. Bostedt, S. Bajt, L. Gumprecht, B. Rudek, B. Erk, C. Schmidt, A. Homke, C. Reich, D. Pietschner, L. Struder, G. Hauser, H. Gorke, J. Ullrich, S. Herrmann, G. Schaller, F. Schopper, H. Soltau, K.-U. Kuhnel, M. Messerschmidt, J. D. Bozek, S. P. Hau-Riege, M. Frank, C. Y. Hampton, R. G. Sierra, D. Starodub, G. J. Williams, J. Hajdu, N. Timneanu, M. M. Seibert, J. Andreasson, A. Rocker, O. Jonsson, M. Svenda, S. Stern, K. Nass, R. Andritschke, C.-D. Schroter, F. Krasniqi, M. Bott, K. E. Schmidt, X. Wang, I. Grotjohann, J. M. Holton, T. R. M. Barends, R. Neutze, S. Marchesini, R. Fromme, S. Schorb, D. Rupp, M. Adolph, T. Gorkhover, I. Andersson, H. Hirsemann, G. Potdevin, H. Graafsma, B. Nilsson, J. C. H. Spence, Femtosecond x-ray protein nanocrystallography, Nature 470 (7332) (2011) 73–77.
URL http://dx.doi.org/10.1038/nature09750

[3] Y. H. Jiang, A. Rudenko, O. Herrwerth, L. Foucar, M. Kurka, K. U. Kühnel, M. Lezius, M. F. Kling, J. van Tilborg, A. Belkacem, K. Ueda, S. Düsterer, R. Treusch, C. D. Schröter, R. Moshammer, J. Ullrich, Ultrafast extreme ultraviolet induced isomerization of acetylene cations, Phys. Rev. Lett. 105 (26) (2010) 263002. doi:10.1103/PhysRevLett.105.263002.

[4] Nokia, QT - Cross-platform application and UI framework [cited 22.02.2011].
URL http://qt.nokia.com

[5] Agilent Technologies, Agilent U1065A – Acqiris High-Speed cPCI Digitizers – DC282 [cited 11.09.2011].
URL http://cp.literature.agilent.com/litweb/pdf/5989-7443EN.pdf

[6] Agilent Technologies, Agilent U1051A – Acqiris TC890 Time-to-Digital Converter [cited 11.09.2011].
URL http://cp.literature.agilent.com/litweb/pdf/5989-7109EN.pdf

[7] Adimec, Adimec Opal-1000 [cited 11.09.2011].
URL http://www.adimec.com/en/Service_Menu/Industrial_camera_products/High_performance_cameras_for_the_machine_vision_applications/OPAL_series_High_speed_CCD_cameras/OPAL-1000

[8] 1stVision Inc., Pulnix TMC-6740CL [cited 11.09.2011].
URL http://www.1stvision.com/cameras/JAI/dataman/TM-TMC-6740CL_revAw.pdf

[9] P. Emma, R. Akre, J. Arthur, R. Bionta, C. Bostedt, J. Bozek, A. Brachmann, P. B. R. Coffee, F. Decker, Y. Ding, D. Dowell, S. Edstrom, A. Fisher, J. Frisch, S. Gilevich, J. Hastings, G. Hays, P. Hering, Z. Huang, R. Iverson, H. Loos, M. Messerschmidt, A. Miahnahri, S. Moeller, H. Nuhn, G. Pile, D. Ratner, J. Rzepiela, D. Schultz, T. Smith, P. Stefan, H. Tompkins, J. Turner, J. Welch, W. White, J. Wu, G. Yocky, J. Galayda, First lasing and operation of an ngstrom-wavelength free-electron laser, Nature Photonics 4 (2010) 641–647.

[10] J. Feldhaus, Flash-the first soft x-ray free electron laser (fel) user facility, Journal of Physics B: Atomic, Molecular and Optical Physics 43 (19) (2010) 194002.
URL http://stacks.iop.org/0953-4075/43/i=19/a=194002

[11] M. Altarelli, R. Brinkmann, M. Chergui, W. Decking, B. Dobson, S. Dsterer, G. Grbel, W. Graeff, H. Graafsma, J. Hajdu, J. Marangos, J. Pflger, H. Redlin, D. Riley, I. Robinson, J. Rossbach, A. Schwarz, K. Tiedtke, T. Tschentscher, I. Vartaniants, H. Wabnitz, H. Weise, R. Wichmann, K. Witte, A. Wolf, M. Wulff, M. Yurkov, XFEL: The European X-Ray Free-Electron Laser. Technical design report, DESY 97 (2006) 2006.

[12] D. Pile, X-rays: First light from SACLA, Nat Photon 5 (8) (2011) 456–457.
URL http://dx.doi.org/10.1038/nphoton.2011.178

[13] SLAC Photon Controls and Data Systems Group, LCLS – pdsdata Reference Manual [cited 12.09.2011].
URL https://confluence.slac.stanford.edu/display/PCDS/pdsdata+Reference+Manual

[14] R. Andritschke, G. Hartner, R. Hartmann, N. Meidinger, L. Struder, Data analysis for characterizing pnccds, in: Nuclear Science Symposium Conference Record, 2008. NSS '08. IEEE, 2008, pp. 2166 –2172. doi:10.1109/NSSMIC.2008.4774781.

[15] R. A. van Engelen, The gSOAP Toolkit for SOAP Web Services and XML-Based Applications [cited 06.08.2011].
URL http://www.cs.fsu.edu/~engelen/soap.html

[16] R. A. van Engelen, K. Gallivan, The gsoap toolkit for web services and peer-to-peer computing networks, proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002) (2002) pages 128–135.

[17] U. Rathmann, Qwt - Qt Widgets for Technical Applications [cited 06.08.2011].
URL http://qwt.sourceforge.net/

[18] I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, N. Buncic, P. Canal, D. Casadei, O. Couet, V. Fine, L. Franco, G. Ganis, A. Gheata, D. G. Maline, M. Goto, J. Iwaszkiewicz, A. Kreshuk, D. M. Segura, R. Maunder, L. Moneta, A. Naumann, E. Offermann, V. Onuchin, S. Panacek, F. Rademakers, P. Russo, M. Tadel, Root – a c++ framework for petabyte data storage, statistical analysis and visualization, Computer Physics Communications 180 (12) (2009) 2499 – 2512, 40 YEARS OF CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures. doi:DOI: 10.1016/j.cpc.2009.08.005.
URL http://www.sciencedirect.com/science/article/pii/S0010465509002550

[19] L. Torvalds, Git - Fast Version Control System [cited 22.02.2011].
URL http://git-scm.com/

[20] The CASS collaboration, CASS – CFEL ASG Software Suite [cited 13.05.2011].
URL http://www.mpi-hd.mpg.de/personalhomes/gitasg/cass

[21] The HDF Group, Hierarchical Data Format - HDF5 [cited 06.08.2011].
URL http://www.hdfgroup.org/HDF5/

[22] U. Köthe, Generic programming for computer vision [cited 06.08.2011].
URL http://hci.iwr.uni-heidelberg.de/vigra/

[23] D. van Heesch, Doxygen – Generate documentation from source code [cited 13.05.2011].
URL http://www.stack.nl/~dimitri/doxygen/manual.html

[24] SLAC Photon Controls and Data Systems Group, Psana User Manual [cited 05.04.2012].
URL https://confluence.slac.stanford.edu/display/PCDS/Psana+User+Manual

[25] SLAC Photon Controls and Data Systems Group, Pyana User Manual [cited 05.04.2012].
URL https://confluence.slac.stanford.edu/display/PCDS/Pyana+User+Manual

[26] A. Barty, T. A. White, M. Weaver, G. Williams, J. Sellberg, J. Feldkamp, cspad image analysis software for the CXI instrument at LCLS [cited 17.04.2012].
URL http://www.desy.de/~barty/cheetah/

[27] L. Torvalds, The Linux Foundation [cited 11.09.2011].
URL http://www.linuxfoundation.org/