QLA User Guide

SPIRE-RAL-DOC-002260

Version 2.2

6th January 2005

S. Guest, H. Bright

1. Introduction to QLA
2. Conventions used in this Help System
3. Instructions for Reporting Problems
4. The QLA Log File
5. Starting QLA
    1. The QLA Application Menu
    2. EGSE Router
6. Packet Receiver
    1. "Playing" data
    2. Data Selection
    3. Data Source
    4. Time selection
7. Packet Dump
8. Packet Viewer
9. Displaying Parameters
    1. Parameter Selection
        1. Selecting Bolometers From Array Images
        2. Selecting from the Parameters Menu
        3. The Selected Parameters Window
    2. Creating Displays
    3. Clock Displays
    4. Scrolling Lists
    5. Time Series Plots
    6. Compress
    7. Follow
    8. Fix
    9. Other Uses of the Mouse
    10. Time Series Properties Panel

- Image Displays

- Saving Parameter Values to a File
- Printing and Screenshots
- Help
- Using the Console

    1. Imports
    2. Startup Script
    3. Starting and Using Processes
    4. Printing the Values of a Parameter
    5. Saving Parameter Values from the QLA Console
    6. Calling the Help Application from the Console
    7. Starting a Time Series Display from the Console
    8. Starting a Clock Display from the Console
    9. Starting a Scroller Display from the Console

- Configuring QLA

    1. Basic
    2. Advanced

- Troubleshooting

    1. Displaying Science Parameters when no Housekeeping Packages are selected in the Simulator
    2. Selecting SID from the Housekeeping Parameters Menu in the Simulator
    3. The Data Displays show a Single Value and then appear to stop
    4. Incorrect Value set for hcss.ccm.factory
    5. No Data Events generated when there is no SPIRE Housekeeping Data in the Telemetry

- APID and SID Quick Look-up
- Product Metadata to FITS Translations
- Pixel Maps

    1. Short Photometer Array
    2. Medium Photometer Array
    3. Long Photometer Array
    4. Short Spectrometer Array
    5. Long Spectrometer Array

- A Note on the Time Format

- The Engineering Simulator

Last revised 6th January 2005 by S.Guest.

Introduction to QLA

The *SPIRE Quick Look Analysis* program, referred to as *QLA* in this help system, is designed to support the SPIRE instrument tests, and takes telemetry packets as input. A simple simulator is also available that can be used to generate packets for testing.

Data fed into the program can be viewed either in packet form, or as individual parameters, using image displays of raw and converted data. These functionalities can be accessed either via the GUIS or the QLA console, which uses the Jython scripting language.

Parameter data can be saved in ASCII and FITS file format, and GUI components can be printed to hardcopy or to image files.

A technical introduction to *QLA* can be found in the description section of "Package herschel.spire.qla" in the Javadoc.

The Engineering Simulator

An engineering simulator has been produced for *QLA* test purposes. This produces simulated packets with the correct format at the correct rate and allows the user to insert values for testing purposes.

To start the simulator the following must be typed at the command prompt (Note that if you are using the router, this must be started first).

```
> Eng_simulator
```

A **Spire Engineering Simulator** window will appear. This consists of a **Selector** menu of radio button options, **SEND** and **SendS** buttons, text boxes to set the timers, and an image which indicates whether data is currently being sent to the router.

Clicking on any of the first four radio buttons (**Full Photometer**, **Full Spectrometer**, **Housekeeping**, or **Alphabetically**) will bring up **Simulator Parameter Selector** windows, which are menus of the SPIRE parameters. The first three are subdivisions of the full list, and the fourth is the entire list in alphabetical order.

When a parameter is selected from the menu a **Simulator Parameter Settings** window for that parameter appears with a slider. Further selections from the same **Simulator Parameter Selector** window will add more parameters to this same **Simulator Parameter Settings** window. Separate windows are generated for science and housekeeping data. If a particular parameter has already been selected, a warning message appears to inform the user of this. Multiple selections can be made by using the shift and control keys.

Closing a **Simulator Parameter Settings window** and closing a **Simulator Parameter Selector** window will not shut down the whole simulator.

Pressing **SEND** will start the packet stream from the simulator to the router for all parameters, both science and housekeeping. The values of the parameters can be changed by moving the sliders or by typing values into the text boxes and pressing the enter key. The image in the simulator will start to animate to indicate that data is being sent. The text of the**SEND** button changes to **PAUSE**, and the text of the **SendS** button changes to **PauseS**. Pressing **Pause** pauses the flow of data to the router for all parameters. Pressing **PauseS** pauses the flow of data for science parameters only.

Pressing **SendS** will start the packet stream from the simulator to the router for science parameters, but not for housekeeping parameters. Once pressed, the text of the button changes to **PauseS**, and pressing it again pauses the flow of data for science parameters.

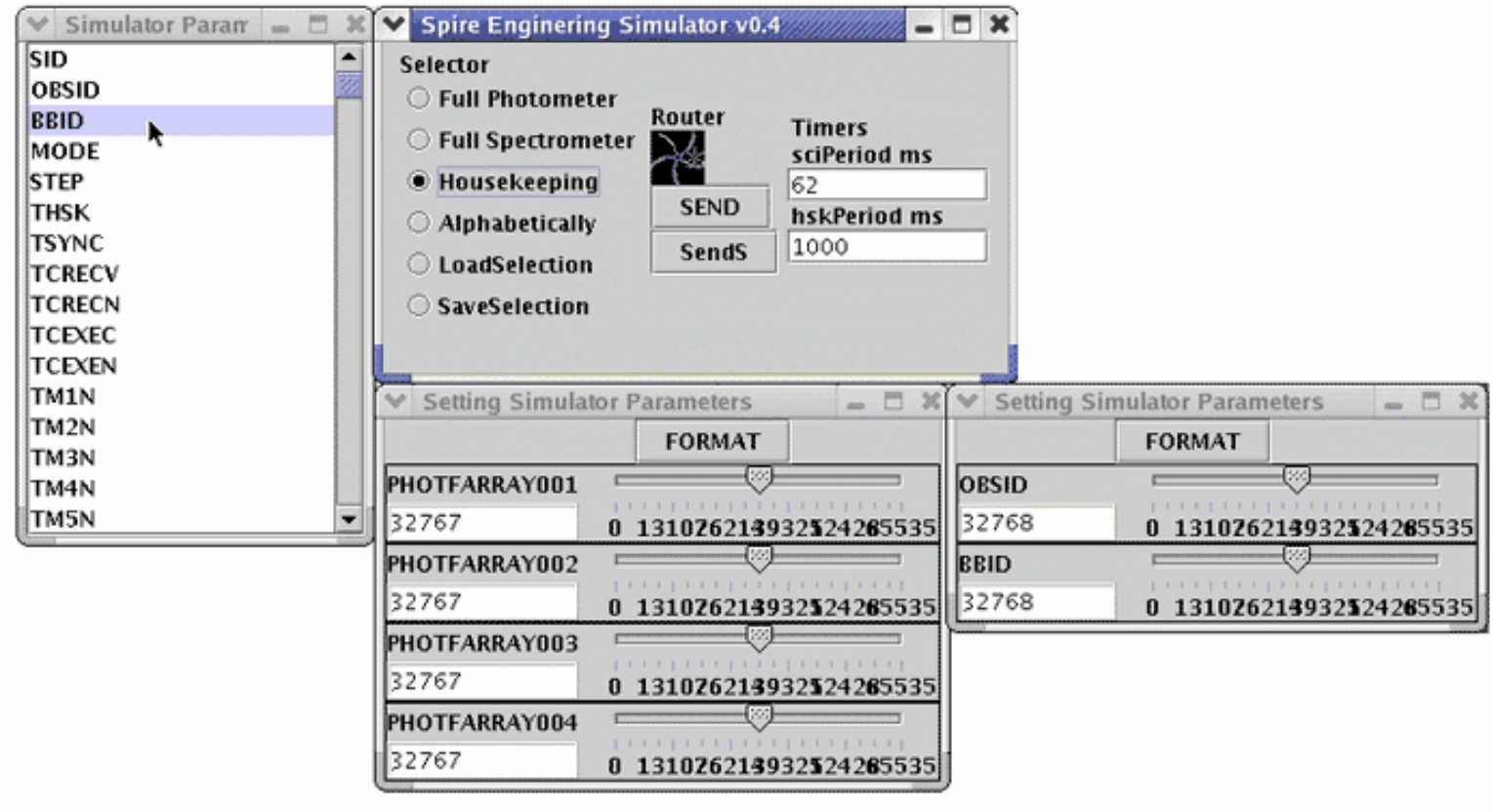
Note that sending data for science parameters without any housekeeping data can cause problems with the displays, as documented in the troubleshooting section.

It is possible to save the currently selected parameters as a file. Clicking on **SaveSelection** will save a file called *saveSelection.txt* in the directory from which you are running the simulator. Clicking **LoadSelection** launches a file chooser window, which you can use to navigate to the directory where you previously saved *saveSelection.txt*. This will load the stored selection. The format of the file is one parameter name per line, the parameter name being the first word on the line. Any other words on the line are treated as comments, as are whole lines beginning with #, eg

```
#comment
PHOTFARRAY034 another comment
PHOTFARRAY004
#comment comment la la la
```

OBSID

It is possible to change the time period only for existing parameters (ie those that have already been selected and are sending data to the router).



The **Spire Engineering Simulator** window, with 6 parameters selected (4 science and 2 housekeeping). The **Simulator Parameter Selector** housekeeping menu is also visible.

Using The Console

The **QlaConsole** window is started by selecting the **QlaConsole** option from the **SPIRE Quick Look Analysis** window. The console allows access to all *QLA* public methods via the Jython scripting language and any Jython command is accepted. Advanced users can consult the Javadoc.

Really advanced users can access any attribute or method regardless of its declaration provided the class is declared to be public. This is controlled by the `python.security.respectJavaAccessibility` property defined in the file *QlaConsole.props*. Currently this is set to false (i.e. allow all access). Full exploitation of this feature requires inspection of the source code.

There are various scripts contained within the *QLA* build. These are located in *herschel/spire/qla/scripts*. If you are using the application bundled in a jar file, see Conventions used in this Help System for information on extracting these. Once you have extracted them, and saved them to your local file system, you can open them in the console and edit them as you wish (see later in this section). The scripts are also available here. - if you have a problem with a script you can check here to see if a later version is available. Information on creating a pipeline script can be found here.

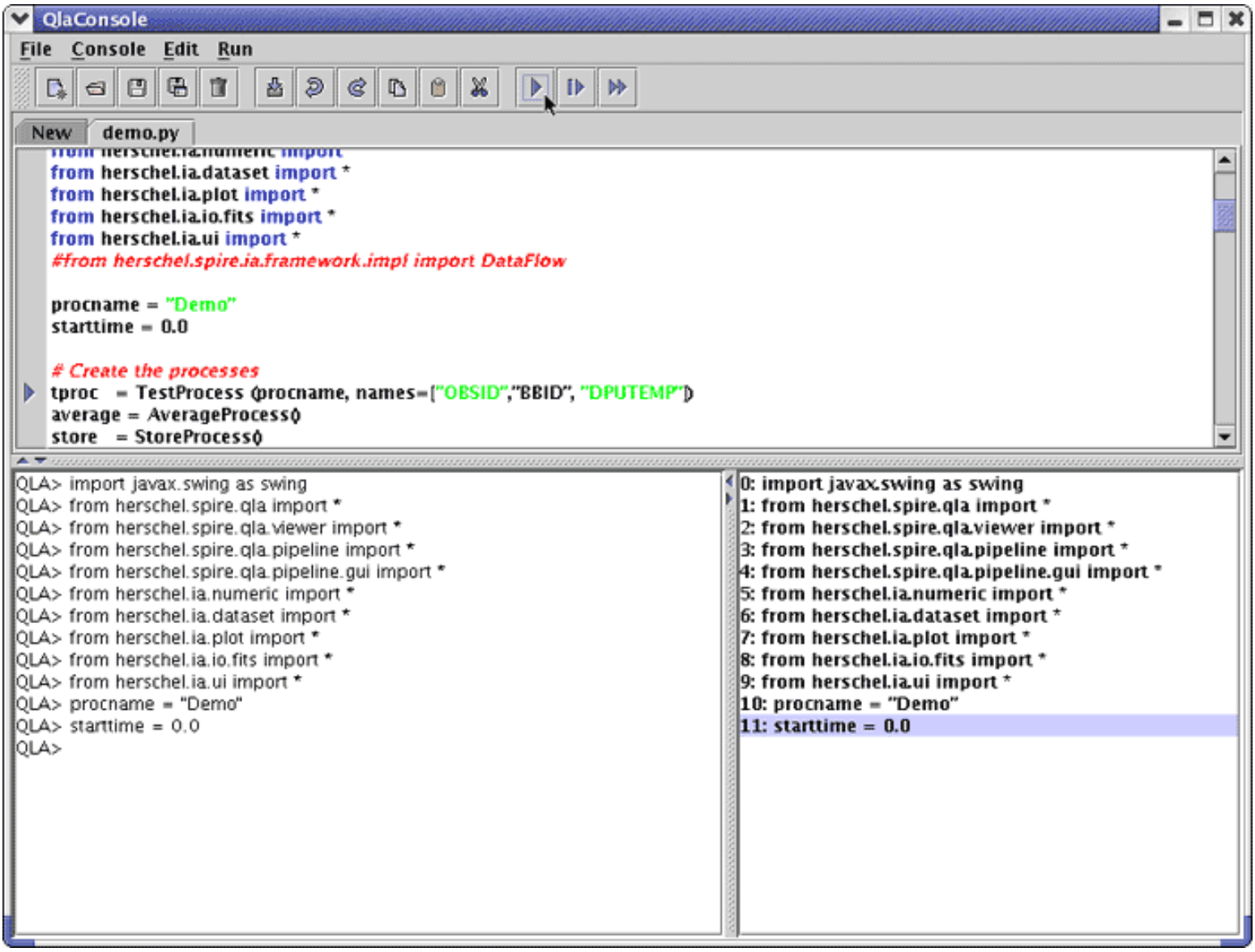
Commands can be typed directly at the console prompt in the command pane (bottom left). Alternatively, prewritten jython scripts can be called from the **Execute**, **Execute line by line** or **Execute in the background** options of the **Console** menu. Calling **Execute line by line** means that all the commands from the scripts will be saved into the console history (bottom right), whereas calling **Execute** saves only one command - **`execfile ("PathToYourScript")`**. Calling **Execute in the background** means that the process will happen in a separate thread, givng priority to non-background processes.

Under the **File** menu there is the option to open a script. If this option is called, the script will be opened in the top pane, but not executed. The user can then edit the script file if they wish to. Clicking in the grey bar on the lefthand side of this pane causes an arrow to appear. If the **Run** menu item under the **Run** menu is pressed, the line of code that the arrow is next to (and any following lines if the command covers multiple lines) will be executed. A highlighted area of text can be executed with the **Run selection** menu item, and the whole script can be run with the **Run all** menu item, both from the **Run** menu. The option is available to save scripts using the **Save** and **Save as...** menu items in the **File** menu. This will save the contents of the top pane.

The right-hand pane of the **QlaConsole** displays the commands that have been executed (the console history), and highlights in red the ones where execution has failed. It is possible to save the successfully executed commands from a session by chosing **Save history** or **Save history as...** from the **Console** menu. Chosing **Import history** from the **Edit** menu will append the current history to the contents of the top pane. Clicking on an executed command in the history pane (successful or unsuccessful) displays an xml version of that command in the command pane.

The **Edit** menu has options for copying and pasting text in the top pane and the command pane, and for undoing and redoing edits to the top pane. Copying and pasting can also be done by highlighting text with the mouse to copy, and pressing the middle mouse button to paste. When in the command pane, the up and down arrows can be used to scroll through previously executed commands, as with a normal operating system console.

The toolbar provides a selection of convenience buttons that fulfil the same functions as some of the menu items mentioned above.



The **QlaConsole** window

Package herschel.spire.qla

This package contains the main API to be used by QLA applications.

See:

Description

Interface Summary	
Converter	Interface to handle conversion
DataListener	Interface for a data listener.
PacketListener	Interface for a packet listener.
ParameterMonitor	Interface for a component that actively monitors parameters.
ParameterSelector	Interface for a component that selects parameters.
Plottable	**Deprecated.** *Use plot package*
SelectionListener	Interface for a parameter selection listener.

Class Summary	
AccumulatedData	This class stores accumulated data for a single parameter.
Controller	QLA control.
DataAccumulator	This class receives packets and extracts the selected parameters.
DataEvent	DataEvent is used to notify interested parties that the available data has changed.
DataModel	This class represents the data as it is received.
DataViewer	Demonstration of DataListener concept.
EventTester	Component to generate (single) events for test purposes.
FunctionConverter	Conversion class using a function expression.
Jython	Jython support class.

Ool	This class handles Out-Of-Limits (OOL) parameters.
PacketEvent	PacketEvent is used to notify interested parties that a packet has arrived.
PacketReceiver	Manage the reception of packets.
PacketType	Class to support operations based on packet type.
PacketUtil	Utility class to interrogate SPIRE packets.
PacketViewer	Packet dump showing complete packet contents.
Parameter	This class allows applications to access data for the parameters they have selected with a `ParameterManager`.
ParameterDescriptor	Utility class for getting information about parameters and extracting parameter data from packets.
ParameterManager	Class to manage parameters for use by data monitors.
PixelMapper	
QLA	QLA main program.
QlaConsole	Interactive QLA Jython-based console (version 2).
QLAPopup	Class that creates a right-click Popup menu.
SelectionEvent	SelectionEvent is used to notify interested parties that parameters have been selected or deselected for monitoring.
SignalConverter	Class to handle signal conversion: raw value <--> converted value
Subsystem	Class to support operations based on subsystem.
Timeline	Class to store a timeline.

Exception Summary	
NoSuchParameterException	Class to generate an exception when given parameters cannot be found in the table, SPIRE_Param_DB.txt.
TableFileNotFoundException	Class to generate an exception when a specified table file cannot be found.
UnmappedVirtualParameterException	Class to generate an exception when given parameters cannot be found in the table, SPIRE_Param_DB.txt.

Package herschel.spire.qla Description

This package contains the main API to be used by QLA applications. It contains classes to receive telemetry packets, extract their parameters, store them with assigned times, and make them available to other applications. It also contains the main QLA program and a Jython-based console, as well as a few demo programs that show how to use the API in a simple way.

The QLA architecture is summarised here.

Introduction to the QLA API

Types of parameters

A common source of confusion is that the word "parameter" is used in a number of different contexts (in Java programming terms you could say that it is "overloaded"). Here is a summary of the different types:

- A **configuration parameter** is one that defines an aspect of the program configuration. It is sometimes also called a **property**. The handling of these parameters is a part of the HCSS (see `Configuration`). The defaults for these parameters are contained in the file QLA.defaults. You can override the default values by creating a file `qla.props` in your home directory and redefining any number of parameters there, for example:

```
        qla.lookandfeel = single
        qla.processes = C:\\Documents and Settings\\sg55\\qla.
processes
        hcss.access.authentication = false
        hcss.access.database = drcu_test@truro
        hcss.ccm.factory = herschel.versant.ccm.
CoreFactoryImpl
```

  Note that this mechanism can be used to override HCSS parameters as well as those specific to the QLA.
- Telemetry packets contain a number of parameters depending on their type e.g. housekeeping packets contain different parameters than science packets. These parameters are defined in the "Sunil table" pointed to by the `qla.parameters` property. This table includes the information required to extract the parameters from the packets, such as bit-offset and bit-length. The programming interface to this information is the `ParameterDescriptor` class.
- A **virtual parameter** is a special type of the above that is not contained in the table. They can be used when it is not known *a priori* which actual parameter to use. Currently these parameters are only used for detector data, where it is possible that either one or all of the detector arrays is switched on. Note that the actual configuration is known when the SID of the science data is known. Virtual parameters are internally mapped to the correct ones on reception of the first new type of science detector (photometer or spectrometer) packet. Any

attempt to call `ParameterDescriptor.getInstance(String)` on one of these parameters before a science packet has arrived, or when different detector science data is arriving will fail with an `UnmappedVirtualParameterException`.

- A higher level of parameter is one that contains the extracted data together with an assigned time as TAI epoch 1958. The normal interface to this data is the `Parameter` class. Objects of this type are created with the help of a `ParameterManager`. Creating a `Parameter` object causes the system to start monitoring that parameter if it wasn't already. A component that does this must implement the `ParameterMonitor` interface. The *name* attribute of these parameters is the same as the one used by `ParameterDescriptor`. Because multiple components might want to look at the same data, a `Parameter` object uses an underlying `AccumulatedData` object. `AccumulatedData` objects are shared between multiple `Parameter` objects if they refer to the same data. A `Parameter` object has two advantages over an `AccumulatedData` one: it has an easier-to-use interface and it maintains its own independent state.

Where to start

All those classes listed above can seem a bit daunting. Here is a quick and by no means comprehensive summary of the most important parts of the API. These are the parts that will normally be used from Jython scripts.

`ParameterManager`	This API controls the registering and deregistering of parameters. Note that the QLA console has a predefined parameter manager variable `pm`.
`Parameter`	This is the API to get at the data.
`ParameterDescriptor`	This is the API for everything you wanted to know about a parameter but were afraid to ask.
`PacketType`	This is the API for everything you wanted to know about a type of packet. Packets are unique by a combination of APID, (service) type, (sevice) subtype and SID
`Controller`	This API serves notification of things happening.

The following HCSS packages are also particularly useful in QLA script development.

`herschel.ia.ui`	General user interface support. `WindowManager` can be used to create windows in a consistent way. It also supports automatic cleanup when a window is closed.

Configuration parameters

These are the configuration parameters used by this package. Subpackages may define their own. These configuration parameters are automatically picked up from the file QLA.defaults. Note that these values can be overridden without changing this file - for details see the HCSS configuration

mechanism.

Parameter Name	Description	Valid Values	Default
qla.lookandfeel	This sets the look and feel to either independent windows (multiple) or windows contained within a single frame (single)	single multiple	multiple
qla.parameters	Name of "Sunil" table file containing parameter descriptions	Filename relative to the location of the QLA.class file	tables/SPIRE_Param_DB.txt
qla.sidtable	Name of table file containing SID descriptions	Filename relative to the location of the QLA.class file	tables/SPIRE_SID_Table.txt
qla.mappings	Name of table file defining mappings from "virtual" detector parameters (e.g. PHOTLW001) to parameters in qla.parameters (e.g. PHOTFARRAY001)	Filename relative to the location of the QLA.class file	tables/Detector_Mappings
qla.processes	Name of the file defining which processes are started automatically when QLA is run and which are added to the "Process Selector" menu	Filename relative to the location of the QLA.class file	qla.processes
qla.apids	This defines the default APIDs that PacketReceiver will request	0-2047 (decimal, not hex) in a comma seperated list delimited by braces	{1280, 1282, 1284, 1285, 1286}
qla.simulatetimes	Turn this on to simulate basic data times if the packets don't contain them	true false	false
qla.convertdets	Turn this on to perform second-stage detector conversion. Currently this will not do anything sensible due to the way the OBS is set up, might change in future	true false	false

qla.buffersize	This defines the initial size of the data buffers	positive integer	10240
qla.packetcache	This is the size of the packet cache used by PacketViewer	positive integer	30000

Example QLA Jython script

Here is a simple example of a Jython script that uses the QLA and Herschel IA APIs to collect some data and write the result to a FITS file. This script can be run from the QlaConsole application. This is a modified version of a real script that was written for a particular purpose. It is provided just for example purposes and does not attempt to exercise all of the QLA API. Note that this script performs no real-time data processing. See the
herschel.spire.qla.pipeline
package for details of how to do this kind of processing.

```
# Import the common Herschel IA stuff that we need
from herschel.ia.numeric import *
from herschel.ia.dataset import *
from herschel.ia.io.fits import *

# Get the names of all the PHOTF array parameters
detectors = filter (lambda x: x.startswith("PHOTFARRAY"),
ParameterDescriptor.getNames(0x0200))

# Monitor some h/k using the built-in ParameterManager. Note that
Parameter objects are returned.
obsp  = pm.add ("OBSID")
bbp   = pm.add ("BBID")

# Monitor the frame time and all the detectors
count = pm.add ("PHOTFFRAMETIME")  # raw data (default)
detp  = pm.add (detectors, 1)      # converted data, all detectors

# This function will be called when the test starts. It's only
purpose is to instruct the QLA
# to call the "end" function when STEP goes to -1 and to remember
the current time.
def start():
  Controller.addStateListener (end, [count.lastTime], "STEP",
0xFFFF)

# This function will be called at the end of the test. Note that
the startTime argument is the
# count.lastTime in the start function above.
def end (startTime):
```

```
  # Save the end time FIRST as data is still coming in...
  endTime = count.lastTime
  obsid = int(obsp.lastRaw)
  bbid  = int(bbp.lastRaw)
  #
  # Create a table dataset and fill in the columns. For DPUCOUNT we
need to convert from double
  # precision to integer (the method of doing so is a little messay
at the moment).
  ds = TableDataset()
  ds["TIME"]     = Column (Double1d (count.getTimes (startTime,
endTime)) - startTime)
  ds["DPUCOUNT"] = Column (Int1D (map (lambda x: int(x), count.
getRaw (startTime, endTime))))
  for i in range(len(detectors)):
      ds["BOL"+str(i+1)] = Column (Double1d (detp[i].getConverted
(startTime, endTime)))
  #
  # Add some metadata. This will map to FITS header records.
  meta = MetaData()
  meta["origin"] = StringParameter ("RAL")
  meta["telescope"] = StringParameter ("HCSS-ILT")
  meta["instrument"] = StringParameter ("SPIRE")
  meta["creationDate"] = DateParameter (Date())
  meta["startDate"] = DateParameter (Timeline.taiToDate (startTime))
  meta["endDate"] = DateParameter (Timeline.taiToDate (endTime))
  meta["obsid"] = LongParameter (obsid)
  meta["bbid"] = LongParameter (bbid)
  #
  # Finally wrap it all in a Product and write it as a FITS file.
  product = Product (meta=meta)
  product.set ("Bolometer data", ds)
  filename = "pht_bol_"+str(obsid)+"_"+str(bbid)+".fits"
  print "Writing file ",filename," with ",str(ds.rowCount)," rows
and ",str(ds.columnCount)," columns"
  fits = FitsArchive()
  #
  # This ensures that the non-standard metadata names (eg obsid)
map correctly to FITS keywords.
  fits.rules.append (DictionarySpire())
  fits.save (filename, product)

# This is the real start of the script. Instruct QLA to kick things
off by calling the "start"
# function when STEP goes to 1.
Controller.addStateListener (start, "STEP", 1)
```

Tutorial

How to write a `ParameterMonitor`

Implementing the
`ParameterMonitor`
interface allows a component to register and deregister for data. If nothing else is already monitoring this data, then registering will cause it to start being monitored. If nothing else is monitoring this data, then deregistering will cause monitoring of that data to stop.

There is a fairly standard way to write a `ParameterMonitor`:

1. Create a `ParameterManager`, normally in the constructor, passing the current instance as an argument
2. Add the desired parameters to the `ParameterManager` and optionally store references to the created `Parameter` objects
3. Provide a method to return the `ParameterManager`, as required by the interface

For Example, to monitor the parameters OBSID and BBID (in Java):

```
class MonitorExample implements ParameterMonitor {
  private ParameterManager pm;
  private Parameter[] params;

  MonitorExample() {
    pm = new ParameterManager (this);
    params = pm.add (new String[] {"OBSID", "BBID"});
  }

  public ParameterManager getParameterManager() {return pm;}
}
```

In Jython, this looks like this:

```
class MonitorExample(ParameterMonitor):
  def __init__(self):
    self.pm = ParameterManager (self);
    pm.add (["OBSID", "BBID"])
  #
  def getParameterManager(self):
    return self.pm
```

Note that if a component is created by the

`WindowManager.add(String,JComponent)`
method (as happens with subclasses of JComponent in the start menu), then there is no need to explicitly deregister parameters - this will happen automatically when the window is closed.

How to write a `DataListener`

A class implements
`DataListener`
in order to be notified of data updates. Note that the rate of these events is controlled by the
`qla.eventrate`
configuration parameter. This means that by default, a
`DataEvent`
is
not
generated on each packet reception. A
`DataEvent`
contains references to
all
the parameters (actually
`AccumulatedData`
objects) that are currently being monitored by
all
active components of the QLA. Note further that a
`DataListener`
is purely passive - it does not cause any parameters to be monitored. The steps involved in writing the listener are:

1. Implement the `DataListener.stateChanged(DataEvent)` method, which is called for each event.
2. Pass the listening instance to the QLA controller by calling the method `Controller.addDataListener(DataListener)`

For example:

```
class DataExample implements DataListener {
  int events;

  DataExample() {
    events = 0;
    Controller.addDataListener (this);
  }

  public void stateChanged (DataEvent event) {
    System.out.println ("Data event "+(++events)+" received at
"+(new Date())+
```

```
                        " from "+event.getSource().getClass().
getName());

    AccumulatedData[] params = event.getData();
    System.out.print ("Parameters in event:");
    for (int i = 0; i < params.length; i++) System.out.print ("
"+params[i].getName());
    System.out.println();
  }
}
```

For the reasons detailed above, a
DataListener
is often also a
ParameterMonitor
, though this is not essential. The two functions work together, as shown by developing the previous examples:

```
class DataMonitorExample implements ParameterMonitor, DataListener {
  private ParameterManager pm;
  private Parameter[] params;

  DataMonitorExample() {
    pm = new ParameterManager (this);
    params = pm.add (new String[] {"OBSID", "BBID"});
    Controller.addDataListener (this);
  }

  public ParameterManager getParameterManager() {return pm;}

  public void stateChanged (DataEvent event) {

    // Print the latest raw value of each parameter
    for (int i = 0; i < params.length; i++)
        System.out.println (params[i].getName()+" = "+params[i].
getLastRaw());
  }
}
```

Here is the same example in Jython:

```
class DataMonitorExample(ParameterMonitor,DataListener):
  def __init__(self):
    self.pm = ParameterManager (self);
    self.params = pm.add (["OBSID", "BBID"]);
    Controller.addDataListener (self);
```

```
  #
  def getParameterManager(self):
    return self.pm
  #
  def stateChanged (self,event):
    # Print the latest raw value of each parameter
    for p in self.params:
        print p.name," = ",p.lastRaw
```

How to write a `PacketListener`

This is fairly starightforward and similar to the data listener case.

1. Implement the `PacketListener.packetReceived(PacketEvent)` method, which is called each time a packet is received.
2. Pass the listening instance to the QLA controller by calling the method `Controller.addPacketListener(PacketListener)`

Here is the actual code from the
`PacketViewer`
demo application:

```
public class PacketViewer extends JPanel implements PacketListener {

  private ScrollingText _text;
  private int packets;

  public PacketViewer() {
    _text = new ScrollingText ("Ready...", 6, 60);
    JScrollPane span = new JScrollPane (_text);
    span.setVerticalScrollBarPolicy (JScrollPane.
VERTICAL_SCROLLBAR_ALWAYS);
    add (span);

    packets = 0;
    Controller.addPacketListener (this);
  }

  public void packetReceived (PacketEvent event) {
    _text.appendLine ("Packet "+(++packets)+" at "+(new Date())+
                      " from "+event.getSource().getClass().getName
());
  }
}
```

Writing a `SelectionListener`

Components that implement this interface are automatically notified whenever there is any change in which parameters are being monitored i.e. whenever a `ParameterMonitor` registers or deregisters for a parameter. Note that if a GUI component is a `ParameterMonitor`, its parameters will be automatically deregistered when it is closed. The steps involved are:

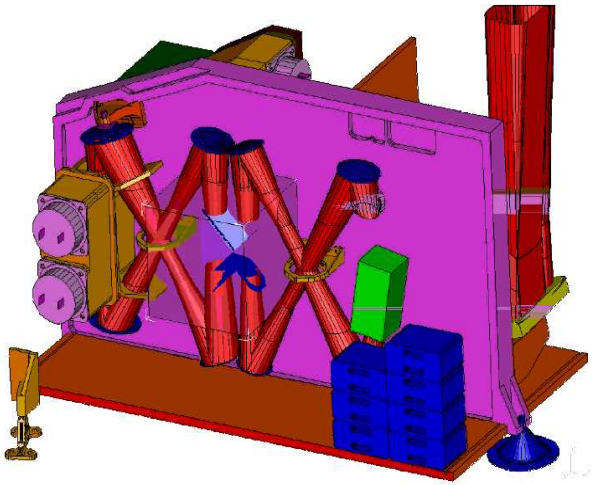
1. Implement the `SelectionListener` interface.
2. Pass the listening instance to the QLA controller by calling the method `Controller.addSelectionListener(SelectionListener)`

`herschel.ia.numeric`	Numeric package. Contains definitions of data types such as `Double1d`
`herschel.ia.numeric.function`	Contains various numeric functions e.g. statistical, interpolation, FFT etc.
`herschel.ia.dataset`	Contains classes to define datasets and products
`herschel.ia.io.fits`	How to read/write products from/to FITS files

SPIRE

Spectral and Photometric Imaging Receiver

SPIRE is one three instruments to be carried on the European Space Agency's **Herschel Space Observatory** (formerly called FIRST). It is designed to make spectral and photometric observations at far infrared and submillimetre wavelengths. It is being built by a European consortium led by the UK, with contributions from France, Italy, USA, Sweden and Spain.



SPIRE Focal Plane Unit, CAD model

The SPIRE Project

- Features
- Science with SPIRE
- Organisation
- Instrument Performance
- Photo Gallery

Related Links

- SPIRE Web site at Cardiff University
- ESA Herschel Science Centre
- ESA Science Communications, Herschel
- RAL Space Science Department
- CLRC
- SAp
- IFSI
- ATC
- ICSTM Astrophysics
- MSSL
- JPL
- SPIRE Web site at University of Lethbridge

Travel Info

- How to get to RAL
- How to get to Cosener's House
- Hotels
- British Airways *(Link)*
- British Midlands *(Link)*
- Eurostar *(Link)*
- Railtrack *(Link)*
- National Express Coaches *(Link)*
- Traffic Information *(Link)*
- Oxford Guide *(Link)*
- The Weather at RAL *(Link)*

If you have any questions relating to the project please contact:

SPIRE Project Office
Tel: +44 (0)1235 446322
Fax: +44 (0)1235 446667

Created by:

K.J. King
k.j.king@rl.ac.uk

Rutherford Appleton Laboratory

Chilton, DIDCOT, Oxon OX11 0QX, U.K.

Last modified - 24th January 2002

Conventions used in this Help System

Sans-serif italic is used for file names, file paths, package names, variable names, method names and application names.
`Fixed-width bold` is used for user input at command lines on the system console.
`Fixed-width plain` is used for user input in files.
Sans-serif bold is used for window names, button names, menu names, menu selections, labels and tab names.

Unless otherwise specified, this document illustrates file paths using Unix file separators: Windows users will have to substitute "/"s with "\"s.

> is used to denote a command prompt. Unless otherwise specified, the commands given in this document are the same for Windows and Unix users.

`QLA>` is used to denote a command prompt in the *QLA* Console

The above conventions may be overidden if the word in question is a link to another section of the help system. If this is the case, the word will be in the usual link style.

Note that file paths quoted in this document are relative to the location of the *qla* directory. This is a subdirectory of the *spire* directory, which is in turn a subdirectory of the *herschel* directory. The location of the *herschel* directory on an individual's file system will vary according to where they installed *QLA*. It may be the case the the files referred to are held in a jar file accessed through the classpath. You can view the contents of a jar file with the following command, where **`foo.jar`** is the name of your jar file:

```
> jar tf foo.jar
```

To extract a particular file to your current directory, for example to copy it to create your own version for configuration purposes (see "Configuring QLA"), use the following command, where **`foo.jar`** is the name of your jar file, and **`foo/bar/fish.file`** is the path to the file you wish to copy within the jar file structure.

```
> jar xf foo.jar foo/bar/fish.file
```

The above jar commads are identical on Windows and Unix.

Instructions for Reporting Problems

Problems detected in the *QLA* should be reported using the SPIRE SPR/SCR reporting system.

Errors detected in this document should be reported to Steve Guest.

The QLA Log File

Every time you run QLA, a file called *qla.log* is created in your home directory. This contains details of Java exceptions that have been thrown, and messages with information or problems. It can be useful for diagnosing or solving a bug with the software.

Starting *QLA*

QLA can be run with two look and feel set-ups, either multiple or single window mode. Screenshots in this document are taken from the multiple window mode. See Configuring *QLA* for instructions on changing the look and feel, and other configurable properties.

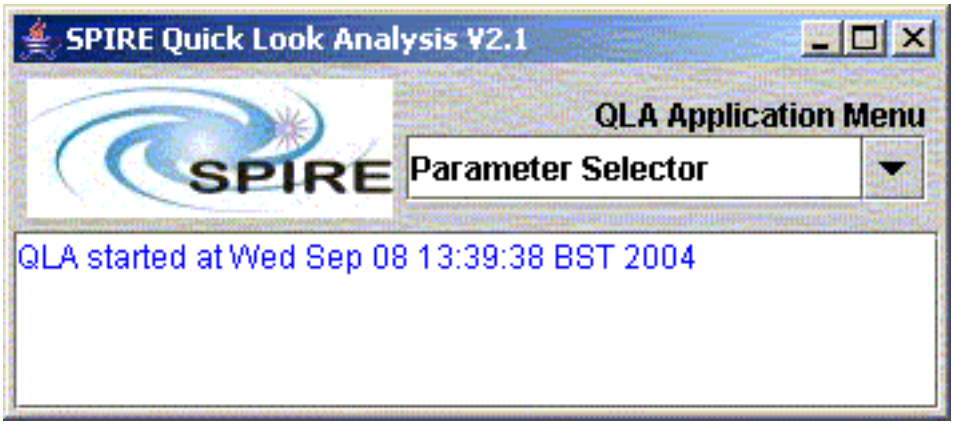
QLA can be started from the command prompt (in any directory) by typing:

```
> qla
```

The **`qla`** command will be set up as either a script or an alias to invoke the correct command.

This brings up two windows if the user is using the default settings. **SPIRE Quick Look Analysis** is the main *QLA* window, which is used to select *QLA* applications. If it is closed it ends the *QLA* session. The **PacketReceiver** window is described here.

In the bottom part of the window, warning and error messages are displayed.



The **SPIRE Quick Look Analysis** window

Some of the QLA program properties can be changed by bringing up a property editor window. To access the property editor, right-click on the window and select "properties". The "Save" button writes updated values to the `QLA.props` file in the `.hcss` subdirectory of the home directory. Note that a change to the look and feel will only apply to subsequent runs of QLA after it has been saved.

The QLA Application Menu

The **QLA Application** menu on the **SPIRE Quick Look Analysis** window is used to select the main functions of the *QLA*. The options are introduced briefly below.

- Parameter Selector: allows the user to select parameters in order to view them as they change over time.
- Image Display: allows the user to view bolometer arrays of raw SPIRE data in the form of an image of the instrument projection on the sky.
- Packet Dump: this dumps the contents of raw telemetry packets to the screen.
- PacketViewer: An application that shows when a packet is received by *QLA*.
- QlaConsole: this allows access to all *QLA* public methods via the Jython scripting language.
- **EventTester**: A developers' application for unit testing aspects of *QLA* functionality.
- DataViewer: A demo application for viewing data values and saving data.
- PacketReceiver: allows the user to select the input data type, source and time period.
- Help Application: launches the help system.

EGSE Router

The EGSE router must be running in order to receive data in near real time. If it is not already running it will need to be started first. Note that the router can be run on a different machine from the *QLA*.

Within the **EGSE Router** selection the first text box displays the host name, the default being **localhost**. Its value is determined by the configurable property `hcss.access.router.host`. The second box displays a four-digit port number, the default being 9877. This is set by the property `hcss.access.router.port`. See the configuration section for details.

If the router is being used this should be started at the command prompt. The four-digit port number forms the last part of the command, and must be the same as that for the `hcss.access.router.host` configurable property as described above. Speak to your system administrator if you have any problems.

```
> java herschel.spire.egse.Router 9877
```

Note: Windows users will need to start another system console in order to do this.

If the router was already running on the specified port on the host machine, an "Address in use" error will be observed. This is not normally a problem.

Packet Receiver

The QLA Packet Receiver is a slightly customised version of the common "access" package's Data Selector tool, see the documentation for that tool.

Herschel Common Science System

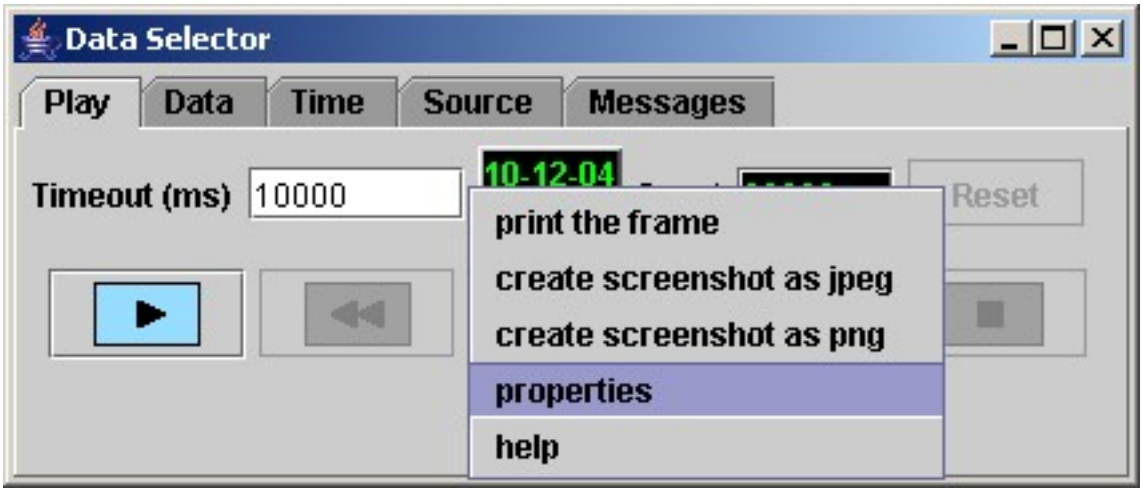
Access Package User Guide

Issue: 1.15, 10th December 2004
Author: Steve Guest

This document describes the usage of the graphical components provided by the access package. The top-level documentation for the package can be found here. A more general "How-To" on accessing the database can be found here.

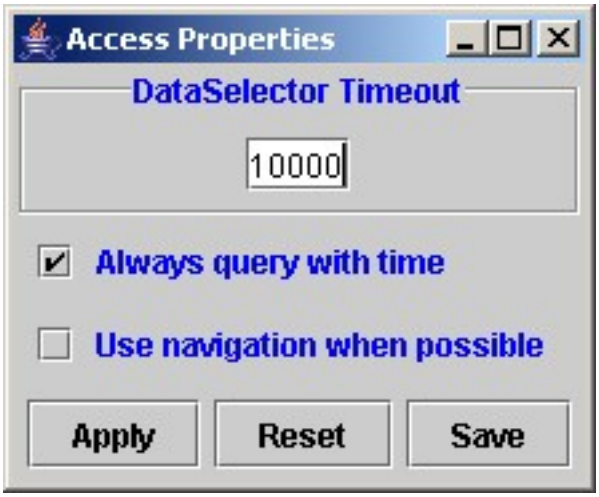
Data Selector

The configuration of the component can be changed by editing its properties, or *user preferences*. Each "tab" has its own associated properties and property editor. To access a property editor, right-click on the window and select "properties".



Selecting the property editor for the "Play" tab

The "Save" button writes updated values to the *application*`.props` file in the `.hcss` subdirectory of the home directory, where *application* is the name of the application e.g. QLA. A tooltip giving extra information is displayed if the mouse is held over a property.

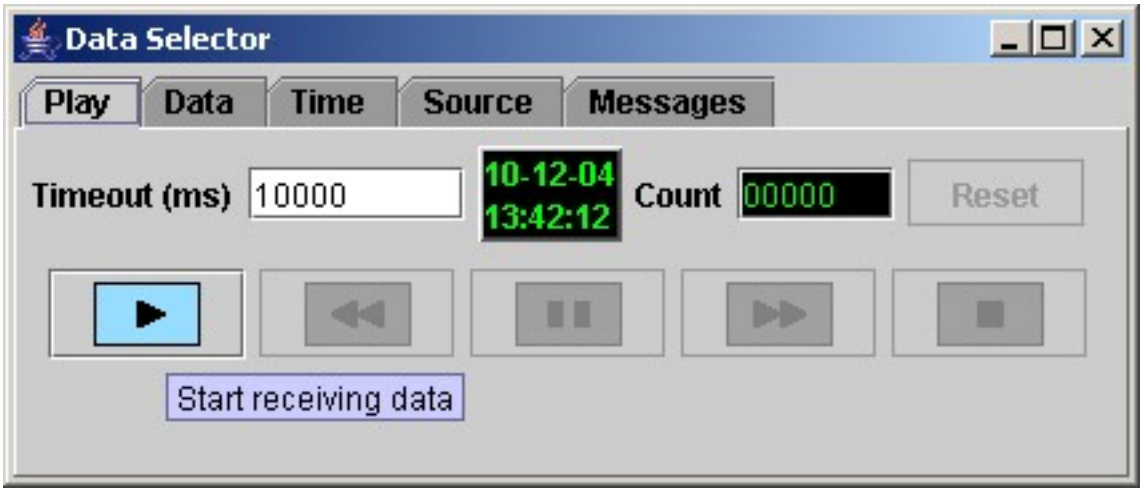


Editing the properties for the "Play" tab

The basic means of operation is to set up the data request using the Data, Time and Source tabs, and then start data reception from the Play tab, which operates in a similar manner to a tape deck. Note that the Data and Time selections are *both* applied.

"Playing" the data

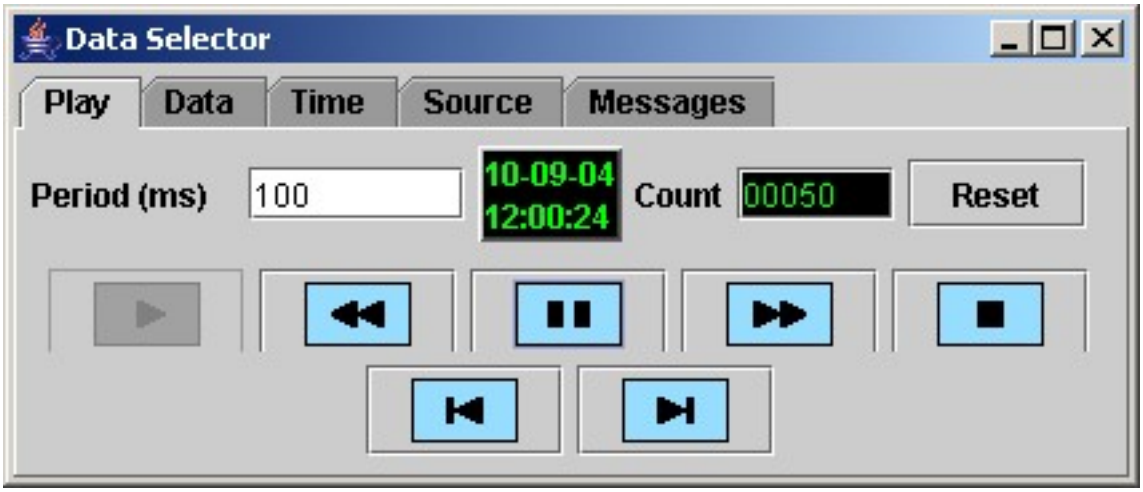
Data reception will start once the **play** button is pressed, under the **Play** tab. Note that if the **play** button is pressed before an input stream is set up (e.g. with a router connection), then a timeout is likely to occur. The **Messages** tab can be used to check if this has occurred.



Selecting the **play** button

When playback mode is selected (i.e. from a database) the speed can be adjusted via the **fast forward** and **fast reverse** recorder buttons on the **Play** tab. In order to use the full features of this mode including backward stepping, ensure that the **cache data** option from the **Source** tab is selected.

The **pause** button pauses playback, and causes two more buttons to become visible, which allow the user to step forwards and step backwards one packet or data frame at a time.

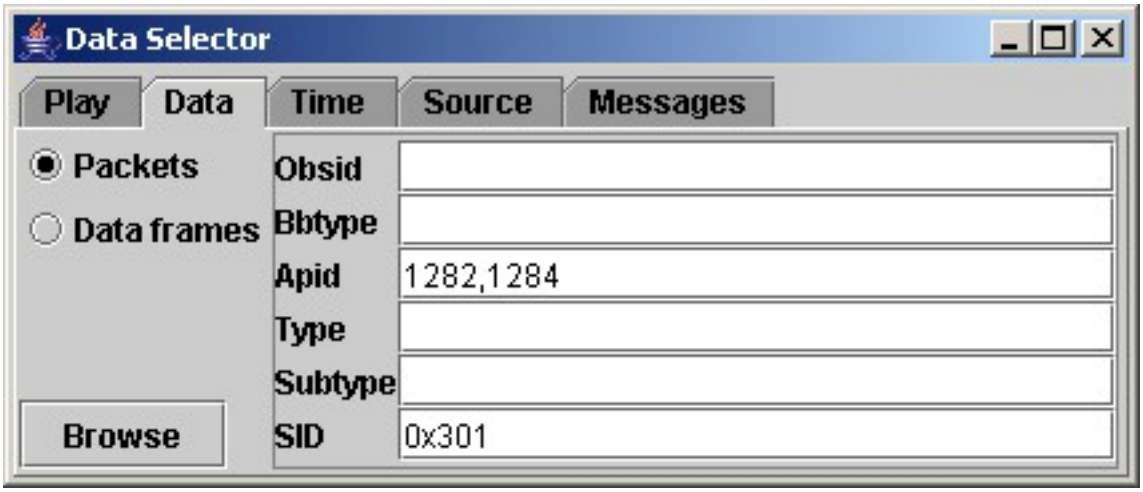


Playback mode, with the pause button having been pressed, and the step forward button being selected

Data Type

The **Data** tab is used to select what data is accessed. Multiple Apids, Types, Subtypes and SIDs can be selected using either commas or white space as delimiters. Numbers can be entered either as decimal or in hex if prepended with "0x". Note that every selection acts as a filter - the default of each parameter is "any". The type, subtype and SID fields are only active when packets are selected. (This might be relaxed at a future time as these fields are valid for SPIRE data frames).

The **Browse** button starts the Test Execution browser, which allows the user to browse the tests which have been carried out. When a test is selected, click on "export selection". When the browser is closed, the times for that test are entered into the "Time" tab of the Data Selector.

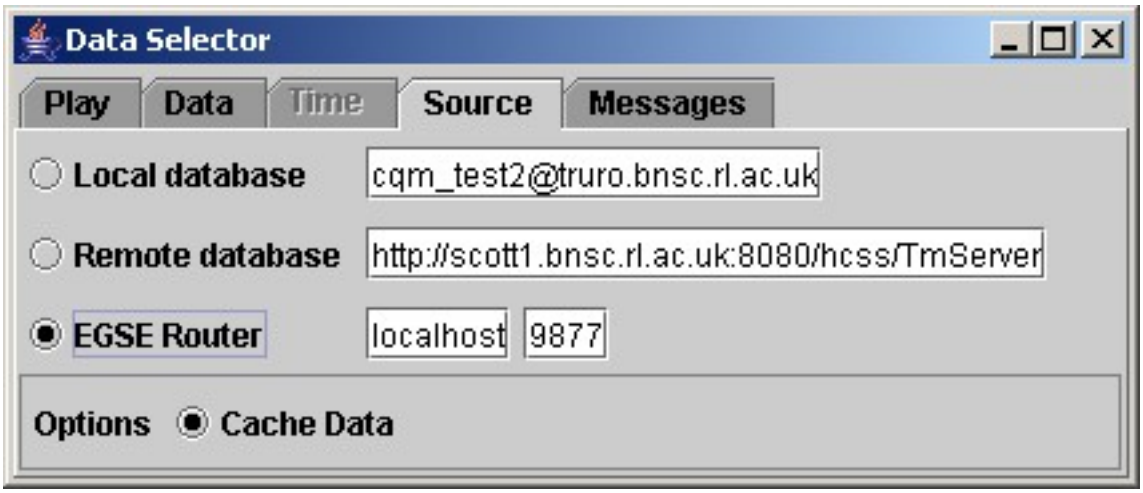


Selecting the data type

Data Source

There are three sources from which data can be retrieved: a local or remote database for playback data, or the EGSE router for real time data. If the **Source** tab is pressed one of the three options can be selected. Note that the default selection can be changed from the property editor window.

When a remote database is selected, the text field displays the URL that will serve the data. *The value of the local database field is passed to the server*. Therefore the name of the remote database is changed in the same way as a local one.



Selecting the data source

A database has to be selected in order to use playback mode. A local database is one that the user is able to access directly from their machine. This is simpler and provides better performance than a remote database, but means that a Versant installation is required on the client side. The text box contains the name of the database and may be edited.

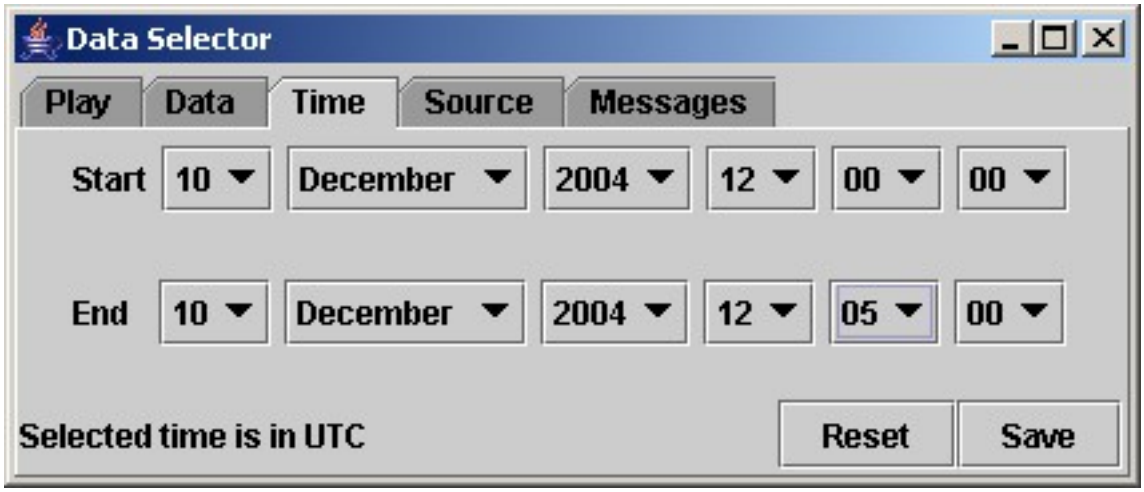
A remote database can be installed anywhere that is network-accessible. Both TCP/IP sockets and HTTP protocols are supported. This does not require a Versant installation on the client, but does require a server to be running. Depending on the selected protocol, either a URL will be displayed (HTTP, the default) or a host name and port number (TCP/IP).

Time

The **Time** tab allows a start and end time to be selected for playback purposes, with each time being entered as a day, month, year, hour, minute and second. Times are specified as by default as UTC. This can be changed if desired from the properties editor.

By default, whenever the start time is advanced, the end time is advanced by the same amount. This behaviour can be changed from the property editor. The "Save" button saves the selected times to the user preferences file, and they will be remembered in subsequent runs of the program. The "Reset" button removes the time as a selection criteria. Visually, the current time is then displayed, but it is not used. This tab is only available if a local or remote database is selected under the Source tab.

Note that the time selection operates as an *additional* filter to anything specified in the Data panel. Use the "Reset" button to clear it.



Selecting the time range

Packet Dump

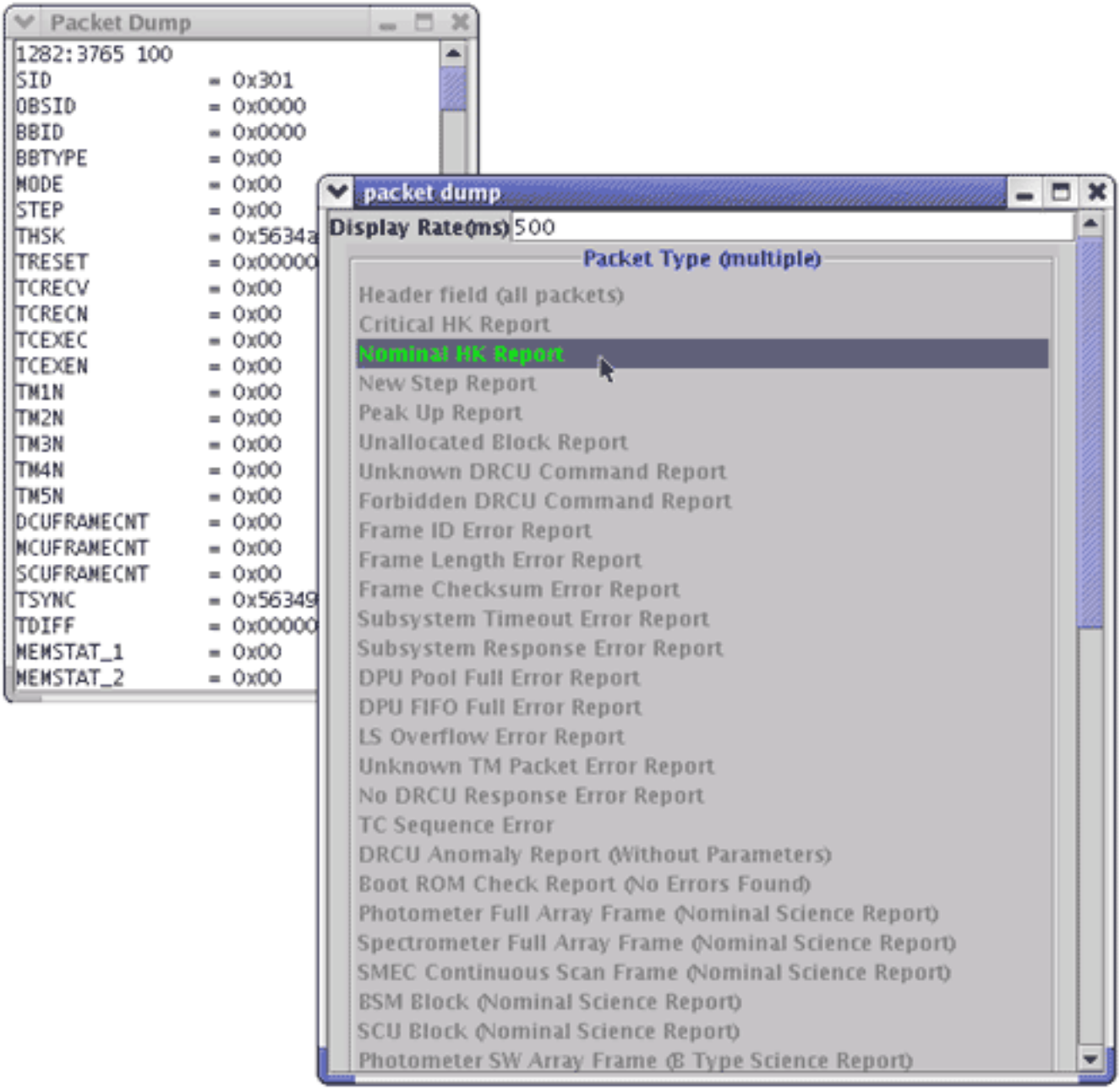
The *PacketDump* application dumps the contents of a certain packet type to the screen.

Clicking the popup menu mouse button (the right mouse button in most operating systems) in the **PacketDump** window brings up a popup menu with a **properties...** option. Selecting this brings up a new window which allows the user to select packets according to their description. The default selected packet is "Nominal Housekeeping". More than one packet can be selected by using the shift and control keys with the mouse button.

The default display update rate is 500 milliseconds, and this can be changed by entering a new value in the **Display Rate** section of the **properties** window. The packet contents can be displayed in hex, decimal or binary, according to the option selected in the **Display in** section of the **properties** window.

All properties can also be changed from the QLA Console application using "set" methods, for example *setDisplayRate*. See the API for details of these methods.

Note that when new packet types are selected, *PacketDump* starts to use the changed values to select packets from the packet stream at once. But, until the **apply** button is pressed, the contents of packets are interpreted according to the properties *before* the changes have been made. Note that currently you also have to press the **apply** button after calling the "set" methods in order to let the change take effect.



The **PacketDump** window and its associated properties window

Packet Viewer

This is an application that shows when a packet is received by QLA. It displays the values in hex, with the offsets on the left.

If the **Pause** button is pressed, the display freezes on the packet currently being received. The **>** and **<** buttons can then be pressed to step through and view the data for individual packets.

Displaying Parameters

In order to watch a parameter evolve with time, a user needs to select the parameter they are interested in, and then select how they wish to view it. Both of these selections are made via the **Parameter Selector** window which is started by selecting the **Parameter Selector** option from the **SPIRE Quick Look Analysis** window.

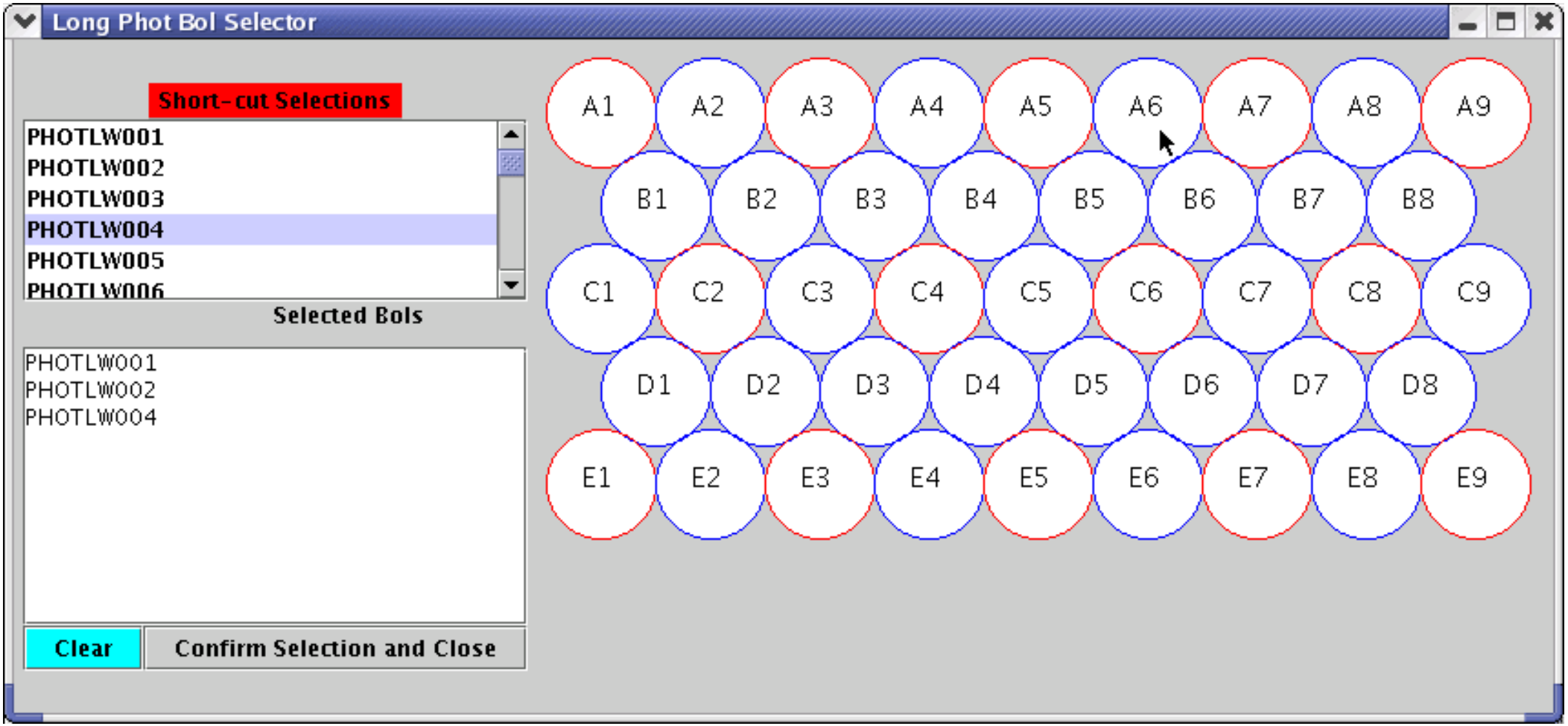
Note that the parameter will start to be monitored from the moment that the display is created for it for the first time. This and all subsequent displays of the parameter will show it's behaviour from this moment.

Parameter Selection

Parameters can be selected in one of two ways. For specific bolometer parameters (PHOTOMETER or FTS), the user can click on the relevant bolometer array name displayed in the Arrays menu, which brings up a window with a visual display from which bolometers can be selected. Alternatively individual parameters can be selected from the Parameters menu, via the relevant submenu.

Selecting Bolometers From Array Images

The **Arrays** menu contains five buttons labelled with the five arrays in SPIRE. Pressing one of these buttons will bring up a window with a display of the bolometer projection on the sky. In this window the top display area shows a menu containing all the bolometer names in the array, and the bottom one displays selected detectors (labelled **Selected Bols**). A mouse click on any bolometer, either in the array layout or upper display area, will put its identifier in the lower display area. A second click on the same bolometer will remove it. The drop-down menu at the top labelled **Short-cut Selections** gives the options to select groups of bolometers. At present only the **Co-aligned** and **All Bolometers** options are implemented. Once selections are completed, the **Confirm Selection and Close** button displays the selected parameters in the **Selected Parameters** area in the **Parameter Selector** window, and closes the window with the bolomoter display. If this button is clicked more than once (if the window with the bolometer display is reopened), extra parameters in the list will be added to the **Selected Parameters** area. Note that parameters no longer in the list will not be removed from the **Selected Parameters** area.



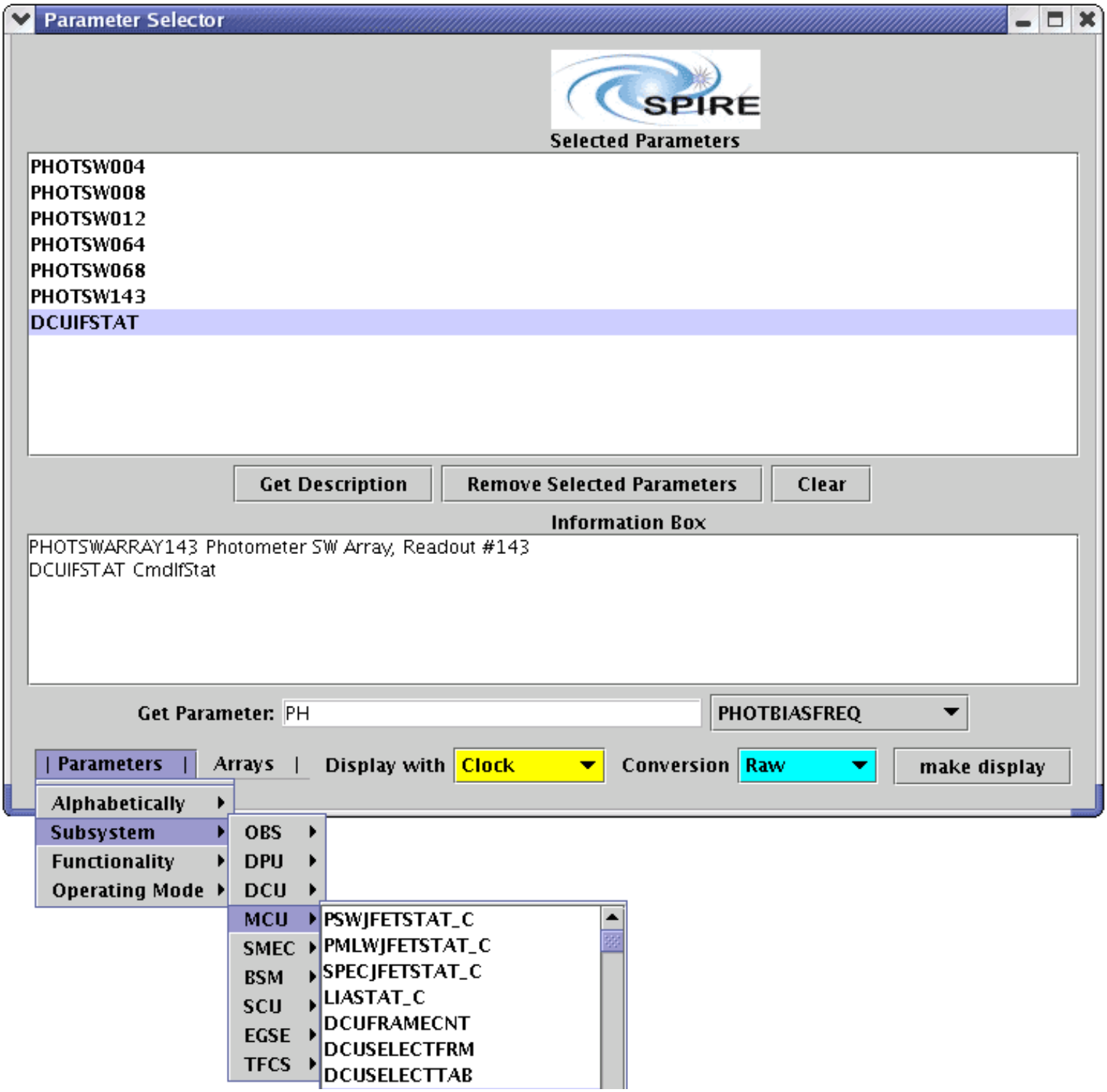
Selecting parameters using a bolometer array display

Selecting from the Parameters Menu

Any parameter can be selected via the pull down **Parameters** menu on the **Parameter Selector** window. At present only the **Alphabetically** and **Subsystem** submenus have been defined and implemented, with the **Functionality** and **Operating Mode** ones still to be defined. The **Alphabetically** submenu contains further submenus which divide the parameters according to their initial letter, and within each of these submenus the parameters are listed in alphanumeric order. The **Subsystem** submenu divides them according to their subsystem type.

Right-clicking on any parameter will display its description in the **Information Box**, as will clicking on the **Get Description** button. This description is based on the entry for the parameter in the *tables/SPIRE_Param_DB.txt* file (See "Configuring QLA").

Parameters can also be selected by typing their name in the **Get Parameter** text box. As you type, suggested parameter names will appear in the drop-down menu on the right. The name that appears at the top of the list is the first alphabetically in the possible parameters available.



Selecting parameters individually via the **Parameters** drop-down menu

The Selected Parameters Window

Once selections of parameters have been made using either the Arrays menu or the Parameters menu, the list in the main **Selected Parameters** area can still be edited. Pressing the **Clear** button, will remove the whole list of parameters from this area. To remove individual parameters, click on them in the list, and then click on the **Remove Selected Parameters** button. Highlighting a parameter and pressing the **Get Description** button displays its description in the information box, in the same manner as right-clicking in the drop down menus (see Selecting from the Parameters Menu).

Creating Displays

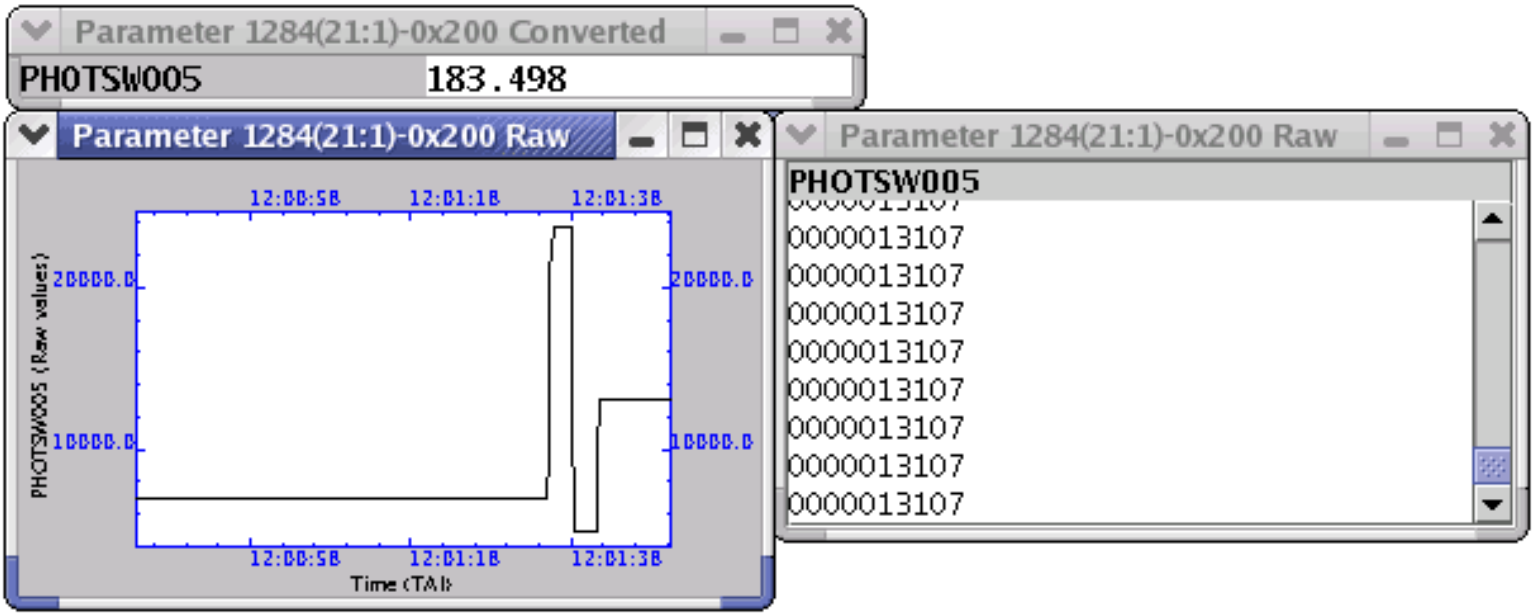
Once parameters have been selected, then the display type: Clock, Scroller, or Time Series is chosen from the **Display using** drop-down menu. The conversion type of the data to be displayed is selected from the **Conversion** drop-down menu - either **Raw**, **Converted**, or **Both**. Clicking on the **make display** button will invoke a new **Parameter Displayer** window of the appropriate display type. The title of the window is determined by the pattern *APID(type, subtype)-SID*. So for example in the figure below the title of the time series window is "Parameter 1284 (21:1)-0x200 Raw".

The selected parameters are viewed as separate components within the window. If the display type is **TimeSeries** and the conversion type is **Both**, two windows containing time series displayers will be launched, one for raw values and one for converted values.

To display a new set of parameters, press the **Clear** button on the **Parameter Selector** window. The already dispatched displays will continue to update. A new set of parameters can then be selected and new display will appear when **make display** is pressed again. Closing the **Parameter Selector** window will not close the **Parameter Displayer** windows invoked by it. There is no limit to the number of times a parameter can be selected or the number of different ways it can be displayed.

If no parameters are selected when **make display** is pressed, then two default parameters: OBSID and PHOTFARRAY001 are displayed.

See "troubleshooting", for a note about displaying science parameters when no housekeeping packages are selected in the simulator.



Parameter Displayer windows for the PHOTSW005 parameter, shown (clockwise from left) as a time series, clock and scroller. Note that the time series and scroller are showing the raw values, and that the clock is showing both raw and converted values.

Clock Displays

The Clock displays the latest values of parameters like a simple digital clock. Each clock will list the parameter names on the left, with a grey background and its value on the right with a white background. If both housekeeping and science values are selected, they will appear on different clock displays. When **Both** is selected from the **Conversion** drop-down menu, the parameter name appears to the left as usual, but there are now 2 values on the right, separated by a forward slash. The raw value is to the left of the slash and the converted value is to the right.

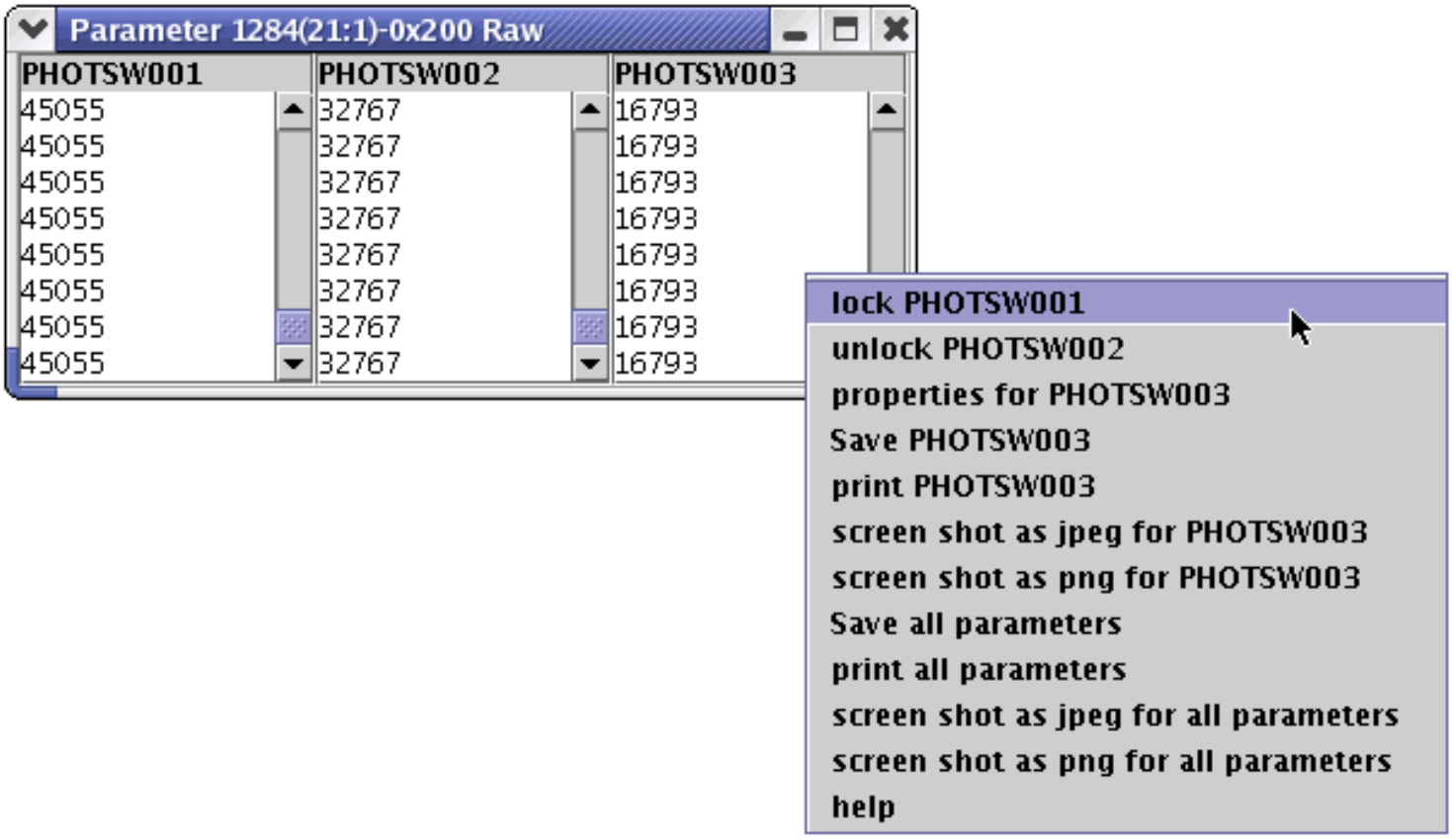
Clicking the popup menu mouse button gives access to some **properties** menu items. If only one parameter is being displayed, there will be just one **properties** menu item, labelled for that particular parameter. If more than one parameter is being displayed, then there will still be an individual **properties** menu item for the parameter clicked on, but there will also be a **properties for all parameters** menu item, which will change the properties of all the parameters being displayed. Clicking this brings up a window with the option to change the number radix of the displayed parameter[s]. The default is decimal, but hex and binary can also be used. This window also allows the user to change the conversion type of the displayed parameter[s].

Scrolling Lists

The Scroller displays the current and past values of the selected parameter in a set of scrollable lists. The lists display all data since the **Select** button on the **Parameter Selector** was pressed. Like the clock displays, the science data and the housekeeping data are displayed in separate windows. When new data arrives, the display automatically switches to show the latest data at the bottom of the list.

If a set of scroller displayers are grouped in one window, different parameters are scrolled independently of each other by default. Clicking the popup menu mouse button on any one of the lists brings up the option to **lock** scrolling with one or more other lists. This means moving one scroll bar will move the scroll bar of all other locked parameters at the same time. Clicking the popup menu mouse button again gives the option to unlock from any previously locked parameters. The format of every line is identical to that of the Clock displayer.

Clicking the popup menu mouse button also brings up the option to set the properties in the same way as for the Clock displayer.



Using the popup menu to select locking and unlocking of scroll bars. In this example the mouse was clicked in the PHOTSW003 parameter window. The PHOTSW003 parameter had previously been locked to the PHOTSW002 parameter, which is why the option to unlock from it is shown here. It had not yet been locked to the PHOTSW001 parameter, so the option to lock to it is shown.

Time Series Plots

The Time Series displays the values of a parameter as a function of time. The display has optional compress mode, follow mode, and fix mode. The user can switch between the different modes by using the options in the right click menu, or by using the mouse, as described in the next sections. The display range and display size can be changed by the user.

The properties panel, accessed via the properties... button on the popup menu, provides a list of preset options for controlling the plot.

Compress

The default mode for the time series display is the compress mode. This keeps the x coordinate of the left edge of the plot constant and compresses the x axis scale when new data become available. This means that new data are always within the display range.

- Double-click the mouse (left button) to return to the default plotting in compress mode after zooming (described in "Other Use of the Mouse"). This mode includes all data from the beginning to the newest data point.
- Control-double-click (left button) to plot in compress mode showing the current x range. This is only relevant if the user has zoomed in or out and changed the default x range (see "Other Use of the Mouse").

Alternatively, click on **full range COMPRESS mode** or **COMPRESS mode** in the popup menu.

The y-axis scale varies to accommodate the highest and lowest data points.

Follow

The follow mode makes the display window follow the latest data point with a constant x coordinate span.

- Shift-double-click to plot in follow mode. The span will continue to display the same width as it had when shift-double-click occurred. In this mode, two red vertical lines are drawn across the upper and lower borders of the drawing box to indicate the x coordinate of the newest data point.

Alternatively, click on **FOLLOW mode** in the popup menu.

The y-axis scale varies to accommodate the highest and lowest data points.

Fix

The fix mode simply fixes the coordinates of all sides of the plot, both x and y axes, so that the displayed part of the data can be inspected without interference of new data. Fix mode only occurs automatically after a particular zoom action has moved the newest data point outside the drawing area (see "Other Use of the Mouse" for details).

Other Use of the Mouse

Click the popup menu mouse button to access the saving options; each data set can be saved separately.

Drag the mouse holding down any button to zoom. The release point of the mouse button can be outside the plotting area, but the press point must be inside it. If the release point is outside the window, this has the effect of zooming out.

The x axis of all the parameter displays is adjusted when the user zooms. However, the adjustment of the y axis when zooming is more complicated. Because several parameters can be displayed at once, a method is needed to determine which parameter display the user intends to update. If the user presses the mouse within a particular parameter display, then this is the display to have its y axis updated. If the user clicks on the grey area either to the left or the right of the white displays, then the parameter display that the click is adjacent to will have its Y axis updated. If the user clicks on the grey area above or below the white display areas, then the top box will have its y axis updated.

If the newest data point is not within range after zooming, plotting will switch into the fix mode to allow inspection of existing data without the interference of changing axes.

Time Series Properties Panel

The properties panel, accessed via the **properties...** button on the popup menu, provides a list of preset options for controlling the plot. A check mark indicates whether or not the property is selected.

If **Y auto range** is selected, it means that the display range of the y axis is automatically calculated when new data is displayed. This should be disabled to keep the y range constant when new data is received.

tick format for X and **tick format for Y** determine the display format of the x and y axes respectively. By default the time (x axis) is displayed in UTC (hh:mm:ss) format. Ay format permitted by

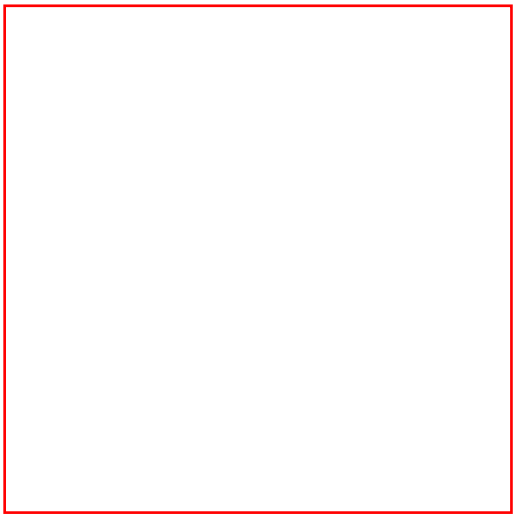


Image Displays

The **Image Display** window is launched when **Image Display** is chosen from the **SPIRE Quick Look Analysis** window.

The user should select the desired bolometer array from the five buttons on the **Image Display** window (e.g. **PHOT SHORT**), which causes *QLA* to automatically register all the bolometer parameters to be monitored. It also launches a new **Imager** window which displays the SPIRE data in the form of an image of the instrument projection on the sky. When a science data event, from the simulator for instance, is received the bolometers with signal will "light up" using a specified colour table, the default being "Heat". The signal values displayed vary from 0 (black) to 16 bit unsigned (full colour). A red circle around a bolometer means it is co-aligned. Either raw or converted data is displayed: this is determined by selecting one or the other from the yellow drop-down menu.

Holding the mouse over a bolometer will display the current signal values (raw and converted) in the information box at the bottom of the imager panel.

Clicking on a bolometer will bring up a time series display window for that parameter.

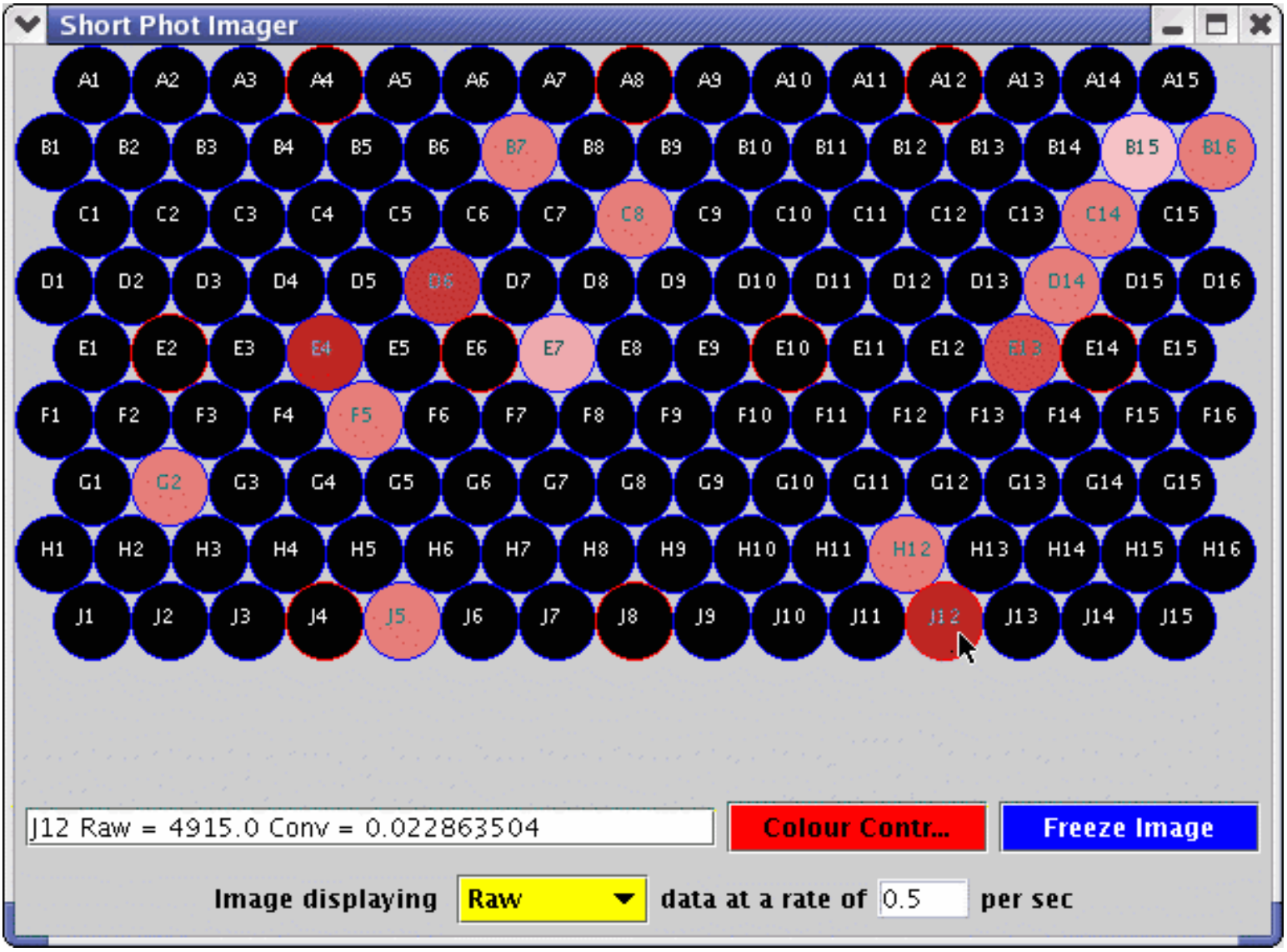
The update rate of the display governs the refresh rate of the image. It has no link to the input rate of data, which is controlled by the qla.eventrate (see Configuring QLA). The value can be set to '0.5' to update the view every 2 seconds.

The **Freeze Image** button can be used to freeze the display at the moment that it is clicked (the text then changes to **Unfreeze Image**). Data updates will not be shown until it is pressed again.

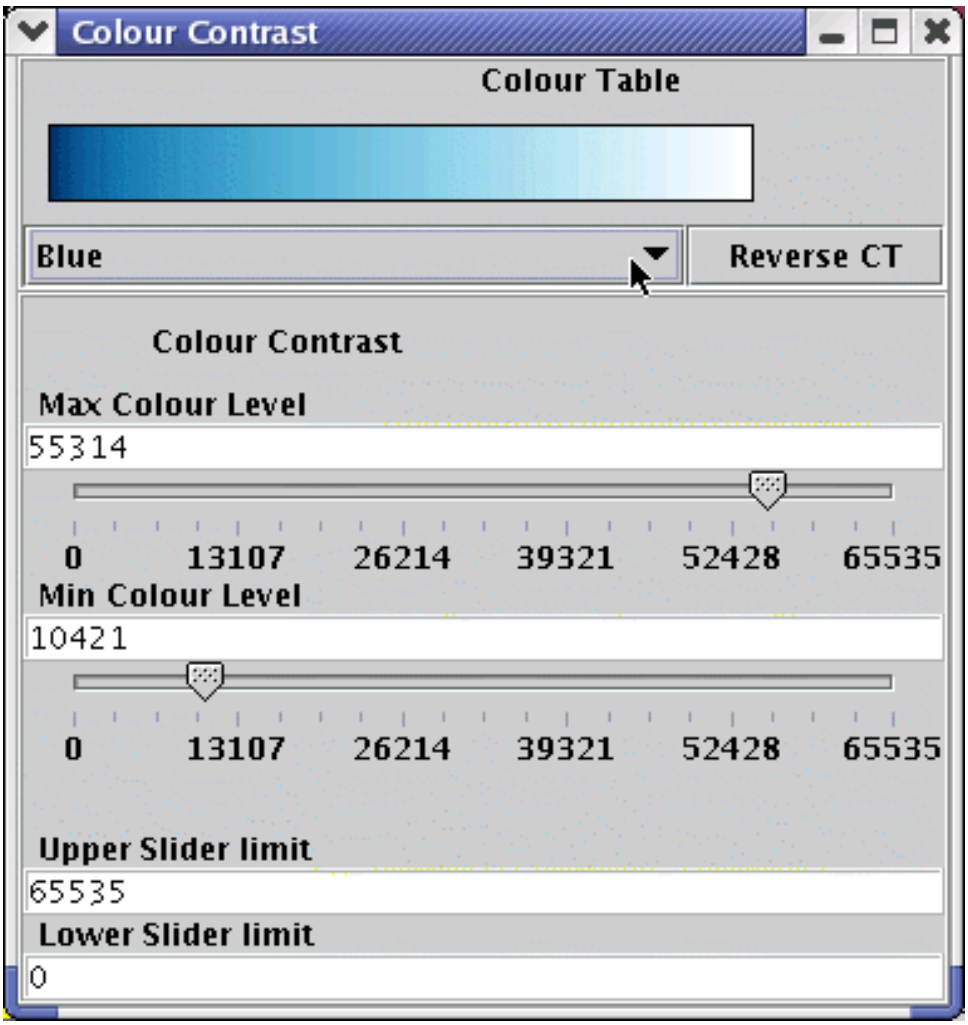
The colour table used to display the data can be edited by clicking on the red **Colour Contr...** button, which opens a **Colour Contrast** window (see the second figure below). The colour table options are "Heat", "Blue" or "Grey", which are chosen from the drop-down menu. The **Reverse CT** button inverts the colour table, so that full-colour indicates the lowest signal value and black the highest.

The colour contrast can be adjusted using the sliding bars and the max and min values for the slider bars can be set in the text boxes below. If the max slider bar value is lower than the signal values coming in, the maximum colour will be displayed for that data. Conversely, if the min slider bar value is higher than the signal values coming in, the minimum colour will be displayed for that data. If the min slider bar value is greater than the max slider bar value, then all bolometers that have a signal value above the minimum value will light up with the same colour, and the rest will remain black.

The max and min values for the slider bars can also be set manually using the text areas at the bottom of the **Colour Contrast** window.



Imager window showing the Short PHOT array. The lighter-coloured parameters are those receiving science events from the simulator. The user is checking the value of J12 by holding the cursor over the relevant bolometer.



Colour Contrast window showing the "Blue" colour table being selected.

Saving Parameter Values to a file

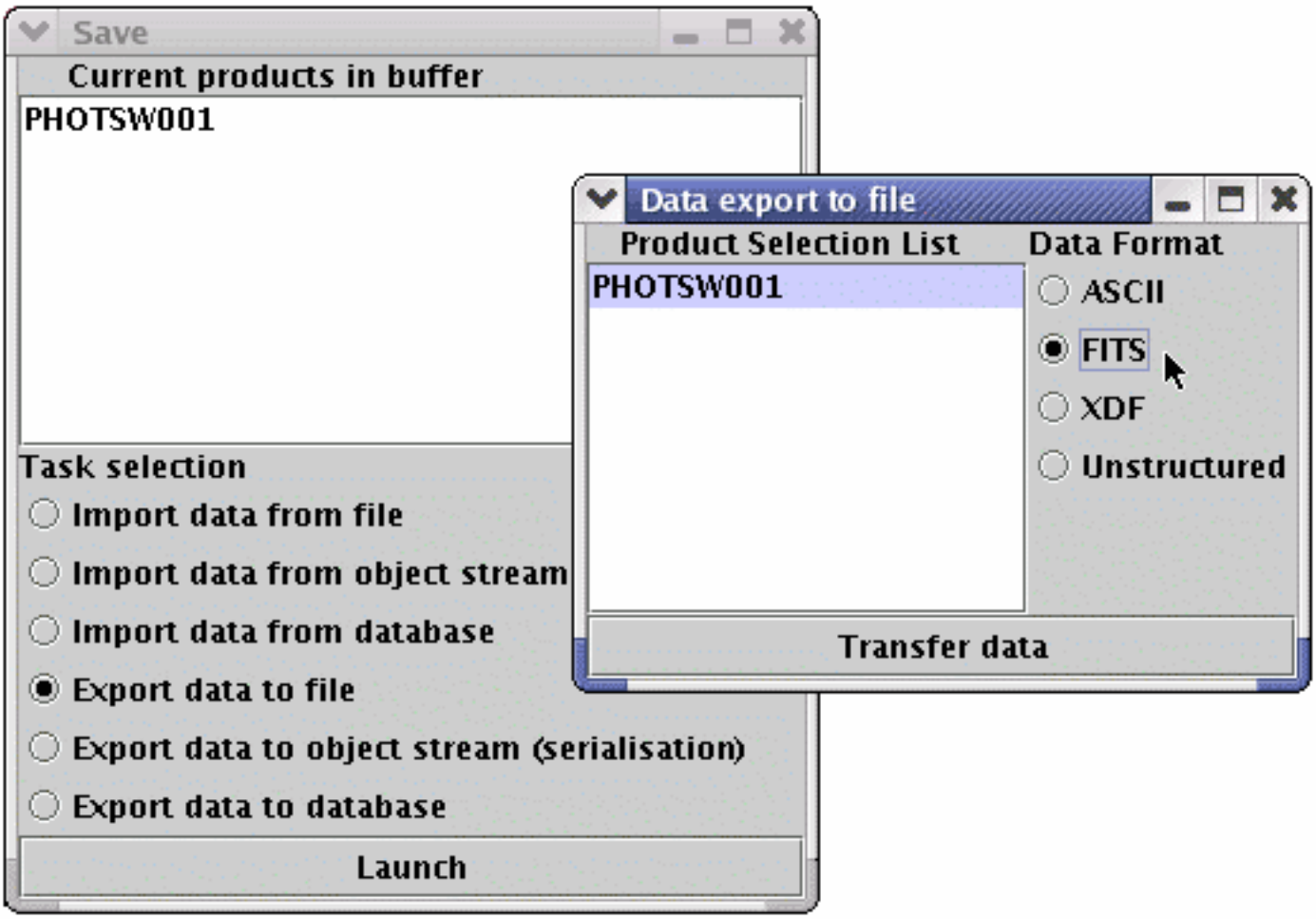
The **Save** window can be accessed from the *DataViewer* demo application (accessed from the **SPIRE Quick Look Analysis** window), or by pressing the popup menu mouse button in *QLA* windows which have a save function implemented. To save data for a particular parameter to file the following instructions should be followed:

- If you are using the *DataViewer* application, enter the parameter name(s) in the **DataViewer** window (note: *QLA* is case-sensitive and all parameter names only use upper case letters) and press the enter key.
- Press **Save** - this brings up the **Save** window showing the parameters in the current data buffer (i.e. the selected parameters that can be saved) and the various options for doing this (note: selecting a particular parameter in this window does nothing).
- Select **Export data to file**, and then click on the **launch** button - this brings up the **Data export to file** window showing the parameters in the current data buffer and the possible file formats that can be used.
- Select which parameter(s) to save from the **Product Selection List**. (Holding down the shift key and clicking on two separate parameters will select these two parameters and all the ones between them in the list. Holding down the control key and clicking on two or more separate parameters selects these parameters only, and not the ones in between).
- Select which data format to use from the **Data Format** selector. Note: **ASCII** refers to a structured flat-file format (i.e. the data is saved in human-readable form but with keywords describing the data); **Unstructured** refers to an unstructured flat-file format (i.e. the data is saved in human-readable form without any structure, e.g. plain columns of data).
- Press **Transfer Data** - this brings up a **FileChooser** window.
- Select the filename for the file to contain the parameter data and press **Save**. Note: If no extension or an inappropriate one for the selected data format is attached to the filename, the default extension for the selected data format is used. The configured values are:

Data format	**Supported extensions**	**Default extension**
ASCII	.dat, .DAT, .txt, .TXT	.txt
FITS	.fits, .FITS, .fit, .FIT	.fits
XDF	.xdf, .XDF	.xdf
Unstructured	.dat	

Not that if the *save* application is launched from the console, or if the **select all parameters**

option is chosen from the clock, scroller or time series displays, the default option is to save all the parameters currently being monitored. See Saving Parameter Values from the QLA Console for details.



The **Save** and **Data export to file** windows

Printing and Screenshots

In most of the *QLA* GUIs the facility is available to print and to create screenshots. Where these options are available, they can be accessed via the popup menu.

If printing is available, the popup menu will either have just one printing option, labelled **print the frame**, or two printing options, labelled **print this component** and **print whole frame**. In the former case, the whole frame will be printed. In the latter case, which is used for composite GUIs, **print this component** will print just the component in question, such as a text area, and **print whole frame** will print the whole frame.

If the option to create a screenshot is available, the option is given either to **create screenshot as jpeg** or to **create screenshot as png**. When selected, these options will generate a file saver dialogue box, so that you can save the generated image file to your system.

Help

QLA incorporates a context-sensitive help system, which is accessed from the GUIs via the popup menu. The page which is opened first depends on the GUI that the help option is called from, and will be the page most relevant to that particular GUI.

The application can also be started from the **Help Application** option in the **SPIRE Quick Look Analysis** window, from the *QLA console*, and independently of *QLA* at the command prompt:

```
> java herschel.spire.qla.Help
```

When the help system is called by any means other than the popup menu, it will open on the first page of the document rather than on a relevant page within the help system. It is possible to use the console to open it on another page, see Calling the Help Application from the QLA Console for details.

Imports

Several Java packages are imported automatically when the console is started. These are: *java.lang, herschel.spire.qla*, *herschel.spire.qla.viewer*, *herschel.spire.qla.selector* and *herschel.spire.qla.dataio.* These packages are defined in the file *QlaConsole.imports*.

There are various ways of importing packages in Jython. A convenient way is to use the **`from`** statement with a wildcard, eg

```
QLA> from herschel.ia.dataset import *
```

However, this form should be used with care, especially in scripts, as it can lead to problems owing to unexpected rebinding of names. It is safer to use the **`import`** statement, eg

```
QLA> import herschel.ia.dataset as dataset.
```

In this case class names have to be prepended with the package variable, in this case **`dataset`**, eg

```
QLA> dataset.ArrayDataset
```

Startup Script

The script *QlaConsole.py* is also executed when the console is started. This defines the following functions (arguments are optional unless they are marked with asterisks):

pd(packets)	Start Packet Dump with the given packet mnemonic, the default if no argument is supplied being nominal housekeeping ("NHK"). The *packets* variable can specified as a single String or as an array of Strings. Examples: **QLA> pd()** **QLA> pd("CHK")** **QLA> pd(["CHK", "PHOTSW", "PHOTMW"])**
pv()	Start Packet Viewer.
clock (names, dataType)	Start a clock display. The *names* variable is the name of a parameter, or an array (or tuple) of names. If *names* is not specified, the parameters known to the console's *ParameterManager* are displayed. For the *dataType* variable, specify 0 for raw data (the default), 1 for converted, or 2 for both. Examples: **QLA> clock()** **QLA> clock("PHOTSW001")** **QLA> clock(["PHOTSW001, "PHOTSW002"], 1)** See Starting a Clock Display for more information.
scroll (names, dataType)	Start a scroller display. The arguments are the same as those specified for clock. Examples: **QLA> scroll()** **QLA> scroll("PHOTSW001")** **QLA> scroll(["PHOTSW001, "PHOTSW002"], 1)** See Starting a Scroller Display for more information.
ts (names, dataType)	Start a time series display. The arguments are the same as those specified for clock. Examples: **QLA> ts()** **QLA> ts("PHOTSW001")** **QLA> ts(["PHOTSW001, "PHOTSW002"], 1)** See Starting a Time Series Display for more information.

`dataio()`	Start the Data I/O application. All monitored parameters are available.
`select()`	Start the Parameter Selector application.
`plot(*p1*, p2)`	Plot one parameter against another. The parameters should have the same sampling rate. If only one argument is supplied, the parameter is plotted against time. Parameters passed to this function can be any of the following: a) Arrays or tuples of numeric values b) Parameter objects c) Parameter names as Strings. In this case the parameters will be registered with the console's ParameterManager (variable *pm*)

This script also looks for a user-defined script and executes it if it exists. This user script must be named *qla.py* and reside in the user's home directory.

Starting and Using Processes

This example shows how to start Packet Dump. It is purely for illustrative purposes, as PacketDump can be started with the "pd" command.

```
QLA> WindowManager.add ("Packet Dump", PacketDump())
```

Printing the Values of a Parameter

There is a pre-defined variable of type *ParameterManager* called *pm* associated with the console. This variable allows full access to the data, including the ability to start monitoring parameters - the paramater is monitored from the moment that **`pm.add()`** is called. Full details are in the Javadoc. It returns a reference to the Parameter, which can then used in conjunction with the **`print`** command to print the parameter value to the screen. The resulting output is shown below the command.

```
QLA> t=pm.add("THSK")
QLA> print t

# PHOTSW001 5 lines, time, raw

1.4480225612358844E9    32767.0

1.448022561297984E9     32767.0

1.448022561360083E9     32767.0

1.4480225614221826E9    32767.0

1.4480225614842818E9    32767.0
```

Note that **`pm.add()`** has an optional second argument: specify **`0`** for raw data, **`1`** for converted data, or **`2`** for both. The default is raw data.

Saving Parameter Values from the QLA Console

For example, to save BBTYPE to a file you would start monitoring it with the parameter manager, and then call the **`save_params()`** command in order to launch the *DataIO* application (see Saving Parameter Values to a file).

```
QLA> pm.add("BBTYPE")
QLA> save_params()
```

The following command does the same thing, except that the data will be saved as IA products

```
QLA> save_products()
```

Calling the Help Application from the Console

The Help Application can be started from the console by typing

```
QLA> Help()
```

If you know the string id of the topic that you wish to open, it is possible to use the **key =** argument to open the help application on a particular page, eg:

```
QLA> Help(key="SelectingInputData")
```

Starting a Time Series Display

To start a time series display from the console use the following function (the arguments are optional):

```
ts (names, dataType)
```

The *names* variable is the name of a parameter, or an array (or tuple) of names. If *names* is not specified, the parameters known to the console's *ParameterManager* are displayed. For the *dataType* variable, specify 0 for raw data (the default), 1 for converted, or 2 for both.

Additionally, you can specify the initial time format for the x axis by using the **`tickFormatX =`** and **`tickFormatY =`** arguments, eg:

```
QLA> ts("PHOTSW001", 0, tickFormatX="UTC", tickFormatY="auto")
```

The above starts a time series of the PHOTSW001 parameter, showing raw data, with the x axis time format as hh:mm:ss, and the y axis format as auto. See Time Series Properties Panel for a description of the time formats available.

It is useful to know that the **`ts`** function returns a reference to a *ParameterDisplayer* object, which is the java class used to display the time series. This can then be used to modify the time format if necessary, using the *setTickFormatX* and *setTickFormatY* methods, eg:

```
QLA> par = ts("PHOTSW001")
QLA> par.setTickFormatX("auto")
QLA> par.setTickFormatY("0.00")
```

Starting a Clock Display

To start a clock display from the console use the following function (the arguments are optional):

`clock (names, dataType)`

The *names* variable is the name of a parameter, or an array (or tuple) of names. If *names* is not specified, the parameters known to the console's *ParameterManager* are displayed. For the *dataType* variable, specify 0 for raw data (the default), 1 for converted, or 2 for both.

Additionally, you can specify the radix for the display by using the **`radix =`** arguments, eg:

```
QLA> clock("PHOTSW001", 0, radix=0)
```

The above starts a clock display of the PHOTSW001 parameter, showing raw data displayed in hex. Specify 0 for hex, 1 for decimal and 3 for binary.

It is useful to know that the **`clock`** function returns a reference to a *ParameterDisplayer* object, which is the java class used to display the clock. This can then be used to modify the radix if necessary, using the *setRadix* method, eg:

```
QLA> par = clock("PHOTSW001")
QLA> par.setRadix(0)
```

Starting a Scroller Display

To start a scroller display from the console use the following function (the arguments are optional):

`scroll (names, dataType)`

The *names* variable is the name of a parameter, or an array (or tuple) of names. If *names* is not specified, the parameters known to the console's *ParameterManager* are displayed. For the *dataType* variable, specify 0 for raw data (the default), 1 for converted, or 2 for both.

Additionally, you can specify the radix for the display by using the **`radix =`** arguments, eg:

```
QLA> scroll("PHOTSW001", 0, radix=0)
```

The above starts a scroller display of the PHOTSW001 parameter, showing raw data displayed in hex. Specify 0 for hex, 1 for decimal and 3 for binary.

It is useful to know that the **`scroll`** function returns a reference to a *ParameterDisplayer* object, which is the java class used to display the scroller. This can then be used to modify the radix if necessary, using the *setRadix* method, eg:

```
QLA> par = scroll("PHOTSW001")
QLA> par.setRadix(0)
```

If you want to see all the data since the parameter started to be monitored, rather than the moment the display is created, use **`fromStart`** argument, eg:

```
QLA> scroll("PHOTSW001", 0, radix=0, fromStart=true)
```

Configuring *QLA*

QLA has a number of configurable parameters that the user can change. Subsystems used by the *QLA* also have configurable parameters; the ones specific to the *QLA* start with `qla`.

Basic Configuation
Advanced Configuration

Basic Configuration

Listed below are *QLA*'s configurable parameters. A default can be overridden by the user if they redefine it in their own configuration file. To do this, they should create or edit an existing file named *QLA.props* in the `.hcss` subdirectory of their home directory.

Note for Windows users: if the `.hcss` directory does not exist, Windows will not let you create it in the usual way. A workaround is to start QLA, right-click on the PacketReceiver window, select "properties", change something in the menu (you might well need to specify the database for eaxample anyway), then press the "Save" button. The directory will then have been created.

Redefinitions can be added with the syntax "parameter = value". For example:

```
spire.qla.lookandfeel = single
```

`spire.qla.lookandfeel` sets whether *QLA* is run with single or multiple windows. The value is either `single` or `multiple`. The multiple window set-up is the default setting.

`spire.qla.parameters`, `spire.qla.sidtable` and `spire.qla.mappings` point to the location of configuration files. If the location of the configuration files is changed (not recommended!), the files must still be in the *QLA* tree or the they will not be loaded correctly. The default values are `tables/SPIRE_Param_DB.txt`, `tables/SPIRE_SID_Table.txt` and `tables/Detector_Mappings.txt` respectively.

`spire.qla.processes` points to the location of another configuration file. Unlike the configuration files described above, this file does not need to be in the *QLA* tree. If the location is not set, or the file not found, *QLA* will use the default file (*spire.qla.processes*). The user can, if they wish, create a *spire.qla.processes* file in their home directory, which will override the file pointed to by this parameter in case of a conflict between the two. The file is used to determine which elements of *QLA* start automatically (by default the **SPIRE Quick Look Analysis** window and the **PacketReceiver** window). In order to set other processes to begin on start-up, the `Start by default` option should be set to `true`. If required, multiple copies of the same application can be started in this manner, by having two entries, both set to `true`. The example below shows how the entry for the Image Display should appear if it is to be started automatically:

```
Selector.BolometerImageDisplayer true true Image Display
```

The normals means to create a customised file would be to copy the default file and modify it.

`spire.qla.tooltips` sets whether or not tooltips are enabled in *QLA*. It can be set to

`true` or `false`, the default value being `true`.

`spire.qla.apids` is the list of APIDs to register for, in decimal format. These appear in the PacketReceiver window's **data** tab on start-up. The default values are `{1280, 1282, 1284, 1285, 1286}`.

`spire.qla.selector.imagemode` is used by the image displays, and is usually set to the `data` value. It can be set to `random` for testing purposes only.

`spire.qla.DisplayRate` is the display update rate in milliseconds of data displays such as clocks, scrollers and time series windows. Note that setting it to zero implies no delay on event reception. The default value is `500`.

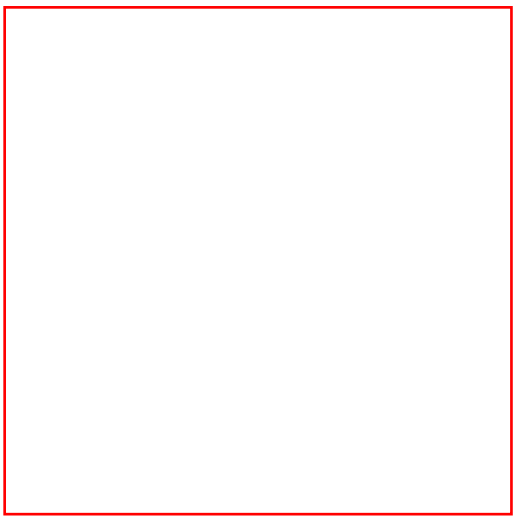
`spire.qla.eventrate` sets the rate of data event dispatching in milliseconds. It can be set to `hk` (the default value), which means the data event is delivered immediately on reception of a housekeeping packet. As housekeeping packets appear every second, this is roughly equivalent to setting the rate to `1000`. However, setting it to `hk` guarantees synchronicity. This may cause problems if there is no housekeeping data in the telemetry. See the Troubleshooting section for details.

`spire.qla.simulatetimes` is normally set to `false`. Setting it to `true` causes it to ignore the input packet times from the simulator and invent it's own. This is used for testing purposes.

`spire.qla.buffersize` is a tuning parameter. This is the initial buffer size used by *QLA* to store data, that is the number of points that can be saved before it has to increase the array size. When buffers are full they have to be copied, which might cause *QLA* to hang briefly. Increasing the buffer size will decrease the frequency of this effect at the expense of greater original memory use. The default value is `10240`.

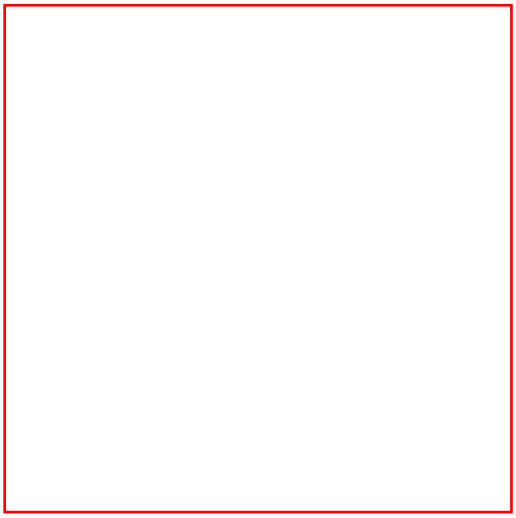
`spire.qla.packetrate` sets the size of the packet cache used by *PacketViewer*. The default value is `30000`.

The data access parameters (the ones starting `hcss.access`) are described at



Advanced Configuration

The *QLA* uses the HCSS configuration mechanism, see



Troubleshooting

This section of the help system helps to solve some common difficulties the user may have when using *QLA*.

Displaying Science Parameters when no Housekeeping Packages are selected in the Simulator

If the user attempts to create a display of a science parameter when no housekeeping packets have been selected from the simulator, *QLA* will print the following to the system console:

```
DataAccumulator: waiting for a h/k packet
```

However the next expected line will not appear:

```
DataAccumulator: Received h/k packet
```

and no display will be created. This is because creating displays involves use of housekeeping packets. However it is possible to view a display under such circumstances, if the developer's tool *EventTester* is started, and the **PacketEvent** button pressed.

Selecting SID from the Housekeeping Parameters Menu in the Simulator

Selecting this parameter at it's default setting will cause the PacketReceiver to print out errors to the command window, and to fail to function correctly. However, the SID can be manually set to a valid value, such as 769, in the simulator by typing it into the text box, and pressing the enter key.

The Data Displays show a Single Value and then appear to stop

This is caused by the packet times not being incremented. The workaround for this is to set `qla.simulatetimes` to be `true` (see "Configuring QLA").

Incorrect Value set for hcss.ccm.factory

If when you start the PacketReceiver you see the following error printed to the command console, it means that the incorrect value has been set for the `hcss.ccm.factory` configurable property:

```
Accessing with params
APID3=1284&APID5=1286&APID2=1282&APID4=1285&NAMEnl.esa.herschel.
versant.ccm.TmSourcePacketImpl&APID1=1280
Got instance of class nl.esa.herschel.access.net.RouterConnection
Error closing stream: java.lang.NullPointerException
java.lang.NoClassDefFoundError: com/versant/trans/CapableWithHash
    at java.lang.ClassLoader.defineClass0(Native Method)
    at java.lang.ClassLoader.defineClass(ClassLoader.java:502)
    at java.security.SecureClassLoader.defineClass
(SecureClassLoader.java:123)
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:250)
    at java.net.URLClassLoader.access$100(URLClassLoader.java:54)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:193)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:186)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:299)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:265)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:255)
    at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:315)
    at nl.esa.herschel.versant.ccm.CoreFactoryImpl.
createInstrumentModel(CoreFactoryImpl.java:180)
    at nl.esa.herschel.access.HcssConnection.getInstrumentModel
(HcssConnection.java:359)
    at nl.esa.herschel.access.net.RouterProductStream.
(RouterProductStream.java:28)
    at nl.esa.herschel.access.net.RouterConnection.openStream
(RouterConnection.java:159)
    at nl.esa.herschel.access.util.DataSelector$OpenHandler.run
(DataSelector.java:299)
    at java.lang.Thread.run(Thread.java:536)
```

The value for `hcss.ccm.factory` should be changed from `nl.esa.herschel.versant.ccm.CoreFactoryImpl` to `herschel.spire.ccm.SimpleCoreFactory`.

The former value only works if you have a Versant installation. See "Configuring QLA" for instructions on setting configurable properties.

No Data Events generated when there is no SPIRE Housekeeping Data in the Telemetry

If no data events are being generated, and there is no SPIRE housekeeping data in the telemetry, you should check the value of the `qla.eventrate` property (see Basic Configuration). Make sure this is not set to `hk`, which guarantees synchronicity of other data events with the SPIRE housekeeping events, but if there is no SPIRE housekeeping data in the telemetry then no data events at all will be generated. Instead, set `qla.eventrate` to a number of milliseconds, such as `1000`. It is possible to check that data events are being correctly generated using the DataViewer tool.

APID & SID Quick Look-up

This is reproduced here purely for convenience - full details are in the *SPIRE Data ICD*. Remember that packets are uniquely defined by a combination of APID, packet type, packet subtype and SID.

Telemetry type	APID (hex)	APID (decimal)
Telecommands, TC verification, Events	500	1280
Housekeeping	502	1282
Photometer Science Data	504	1284
Spectrometer Science Data	506	1286
BSM & SMEC	508	1288
Test Facility Control System	7f4	2036
Test FTS	7f5	2037
CDMS Simulator	7f6	2038

Telemetry type	APID (hex)	Type (decimal)	Subtype (decimal)	SID (hex)	SID (decimal)
Nominal H/K	502	3	25	0301	769
Critical H/K	502	3	25	0300	768
Photometer full array	504	21	1	0200	512
Photometer SW	504	21	2	0102	258
Photometer MW	504	21	2	0103	259
Photometer LW	504	21	2	0104	260
Phot full test pattern	504	21	3	0309	777
Phot SW test pattern	504	21	3	030a	778
Phot MW test pattern	504	21	3	030b	779

Phot LW test pattern	504	21	3	030c	780
Photometer offsets	504	21	4	0207	519
Spectrometer full array	506	21	1	0201	513
Spectrometer SW	506	21	2	0105	261
Spectrometer LW	506	21	2	0106	262
Spec full test pattern	506	21	3	030d	781
Spec SW test pattern	506	21	3	030e	782
Spec LW test pattern	506	21	3	030f	783
Spectrometer offsets	506	21	4	0208	520
BSM Nominal	508	21	1	0612	1554
SCU Nominal	508	21	1	0a20	2592
SCU test pattern	508	21	3	1121	4385
SMEC scan	508	21	1	0410	1040
SMEC selected	508	21	2	0f00	3840
MCU engineering	508	21	3	0814	2068
MCU test pattern	508	21	3	0915	2325
Transparent data	508	21	3	ff00	65280
HK Packet defn	508	21	4	0209	521
HK Table Contents	508	21	4	020a	522
TFCS H/K	07f4	3	25	0100	256
Test FTS H/K	07f5	3	25	0301	768
Test FTS Science	07f5	21	1	002a	42

Product Metadata to FITS Translations

When saving product data after running a script (see BLAH for details on scripts), the following translations are used between the product metadata and the FITS headers.

Product Metadata	FITS Header
origin	ORIGIN
obsid	OBSID
bbid	BBID
fileOrigin	FILEORIG
codeVersion	CODE_VER
samples	NSAMPLES
bolArray	BOLARRAY
bolNum	BOLNUM
bolCount	BOLCOUNT
filetype	FILETYPE
airtemp	AIRTEMP
bbtemp	BBTEMP
pressure	PRESSURE
humidity	HUMIDITY
fts_samples	NSAMPLES
sampleFrequency	ROTFREQU
sampleTime	SAMPTIME
ftstype	FTSTYPE

See the HCSS documentation for information on the translations which are inherited from the HCSS software.

Pixel Maps

Short Photometer Array

Medium Photometer Array

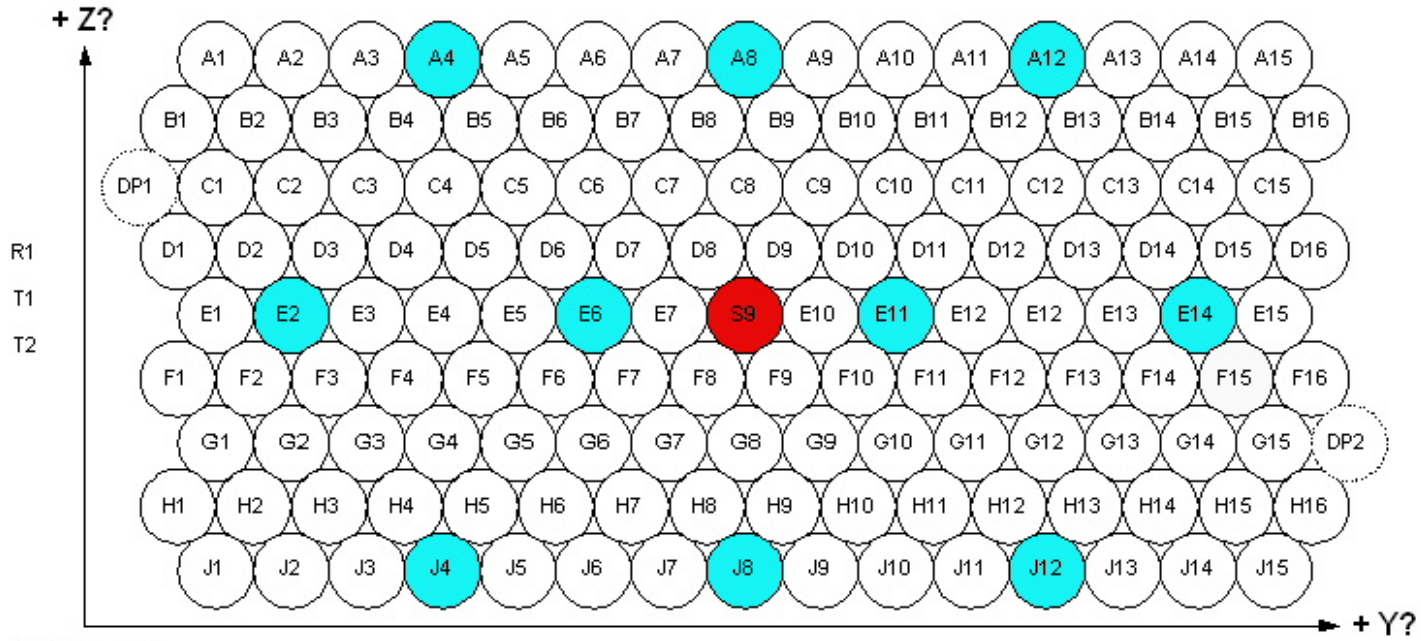
Long Photometer Array

Short Spectrometer Array

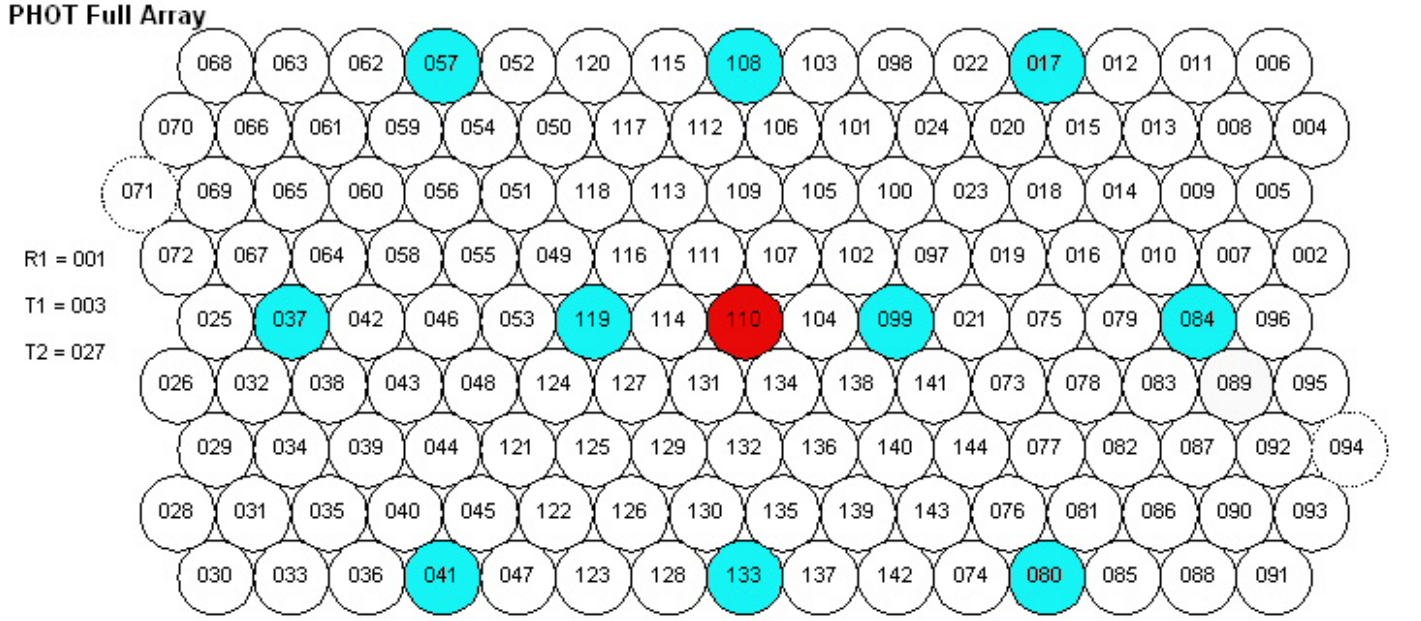
Long Spectrometer Array

Short Photometer Array

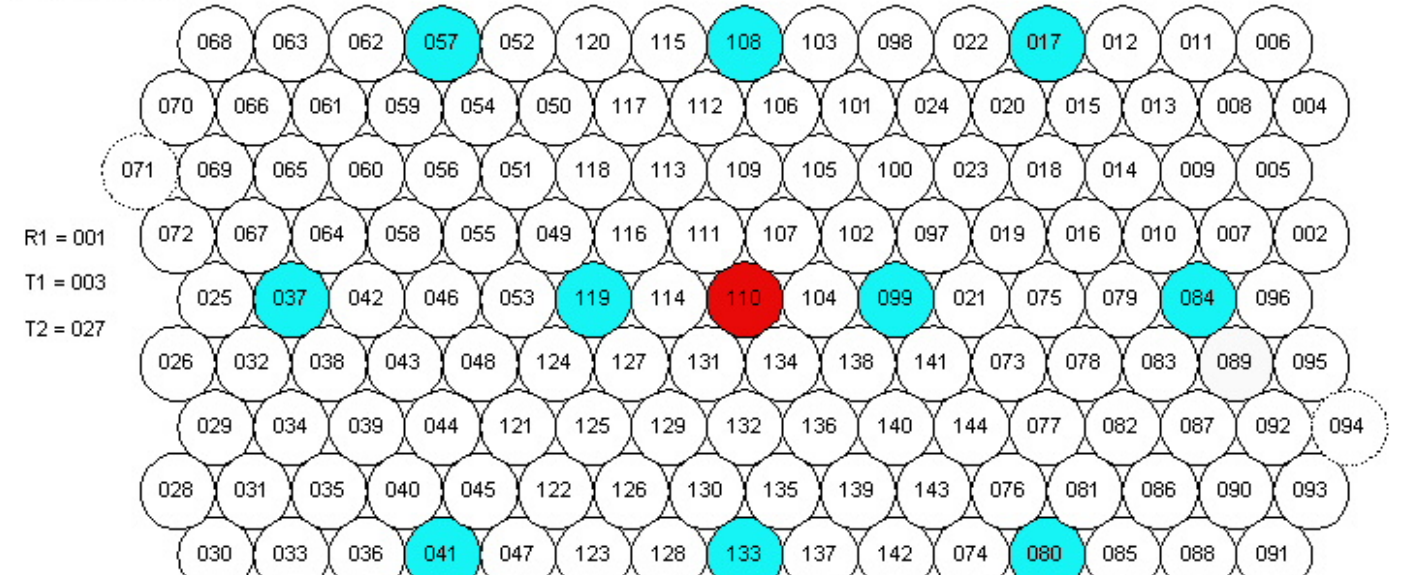
JPL Drawing

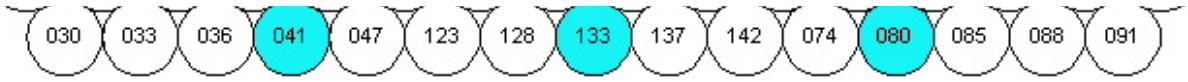


PHOT Full Array

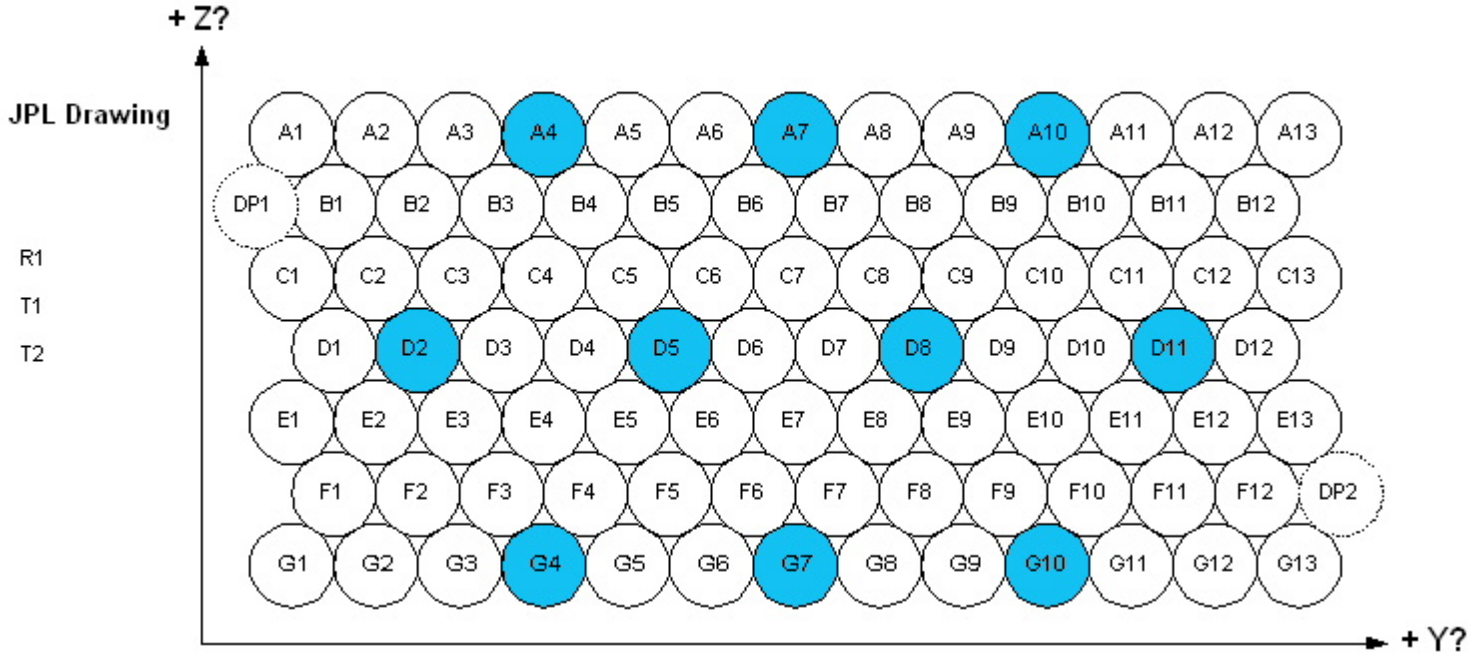


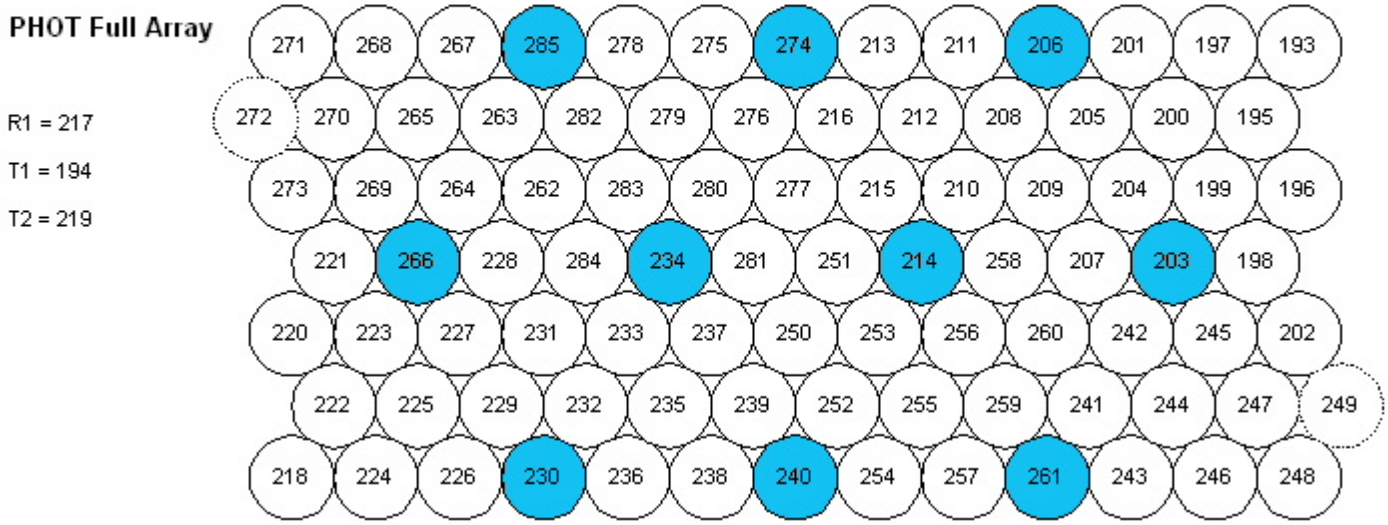
PHOT SW Array

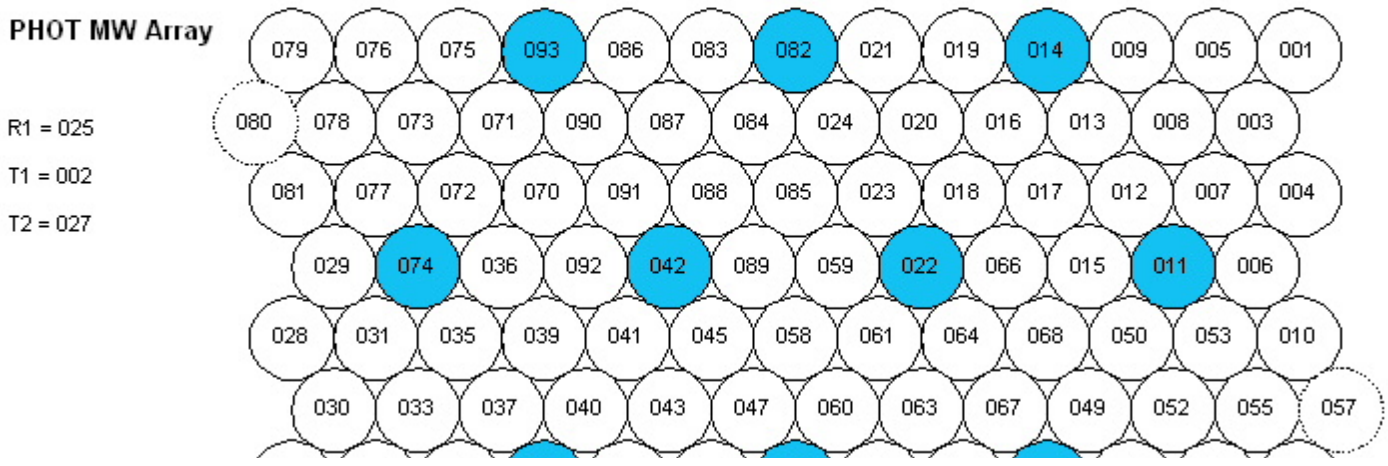


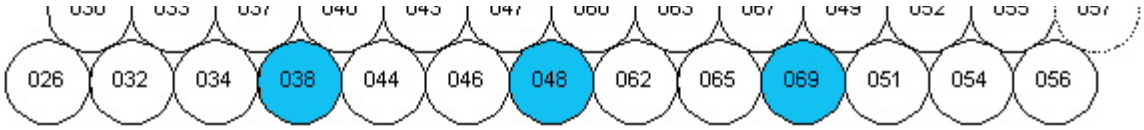

030
033
036
041
047
123
128
133
137
142
074
080
085
088
091

Medium Photometer Array

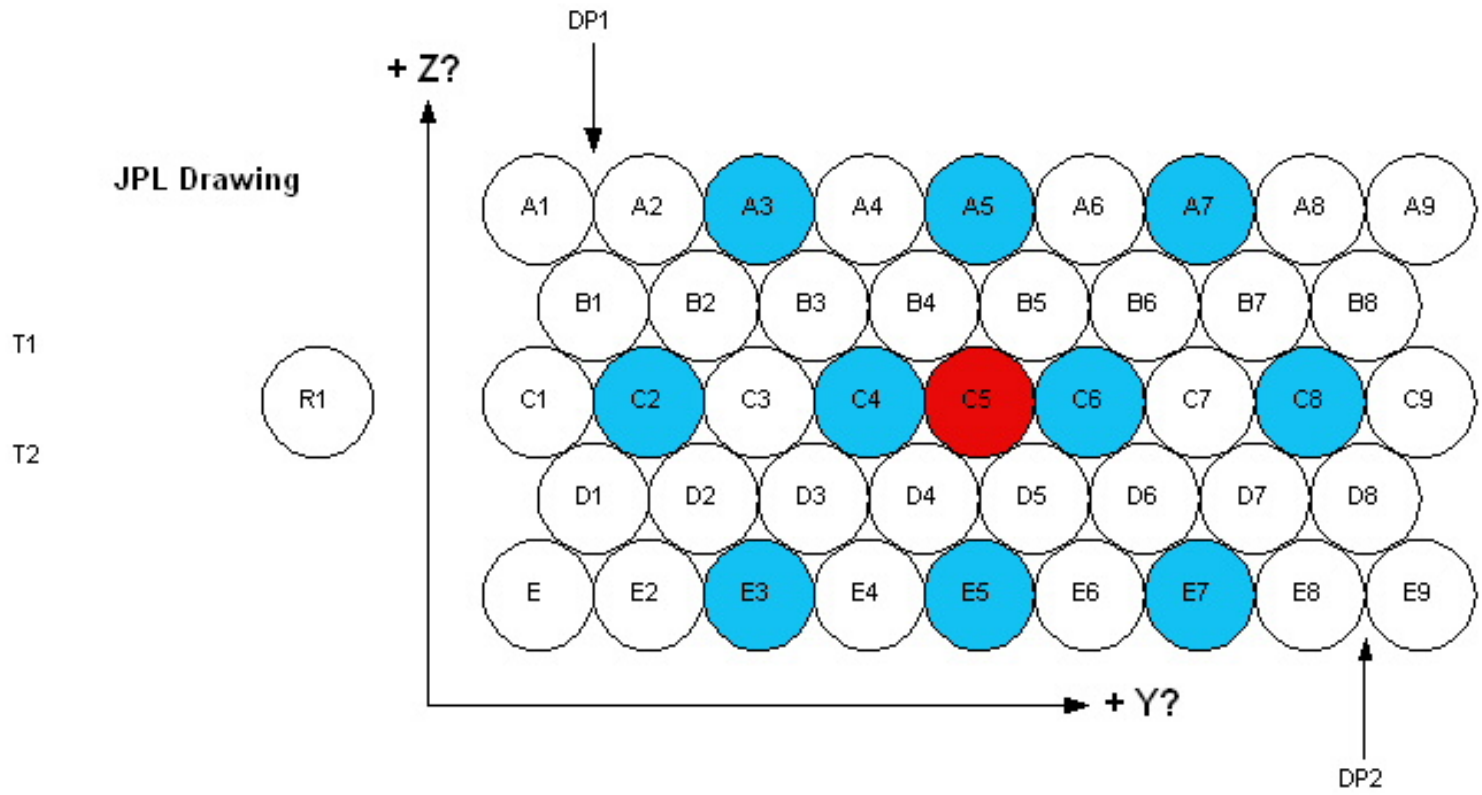


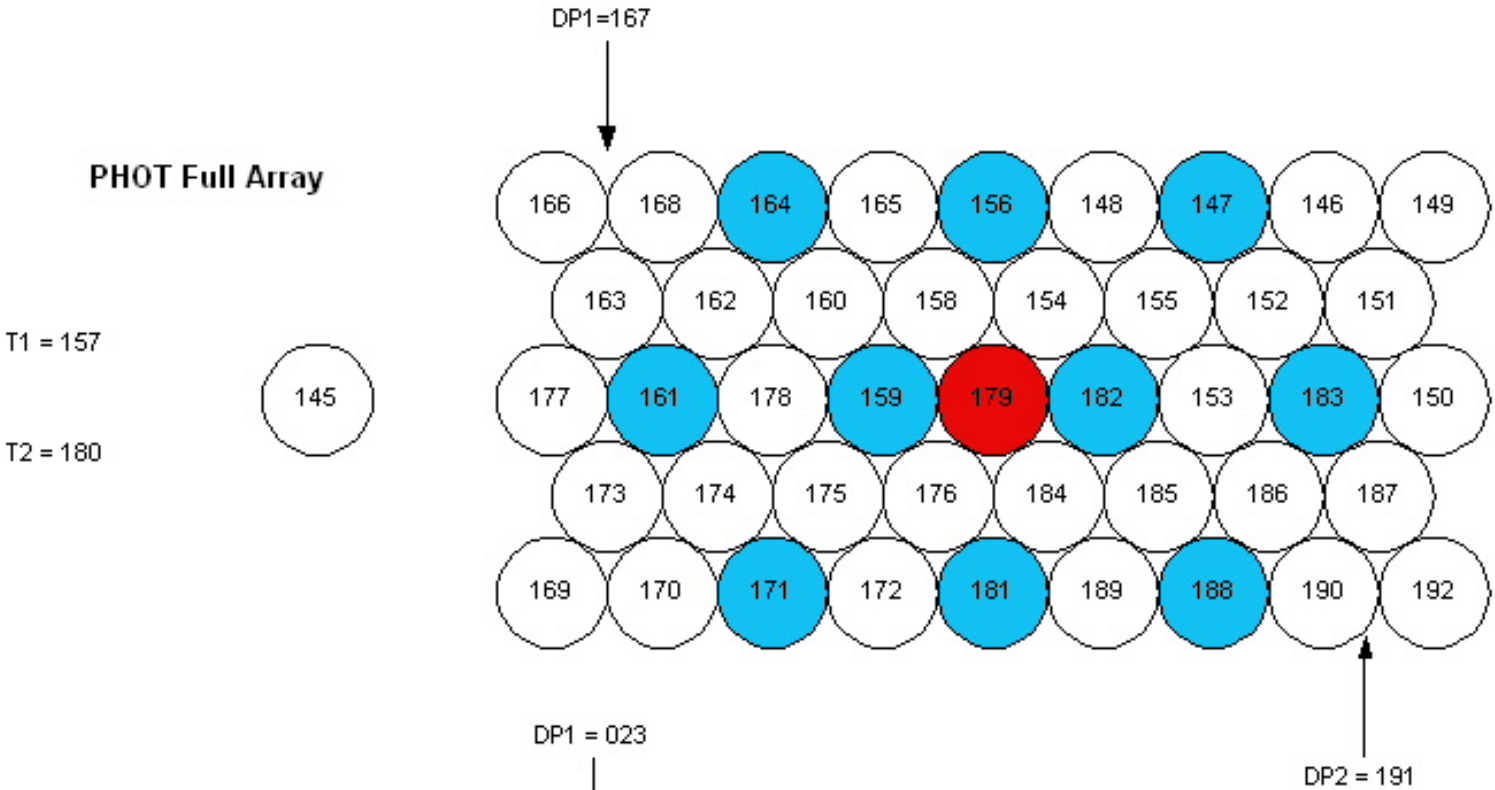


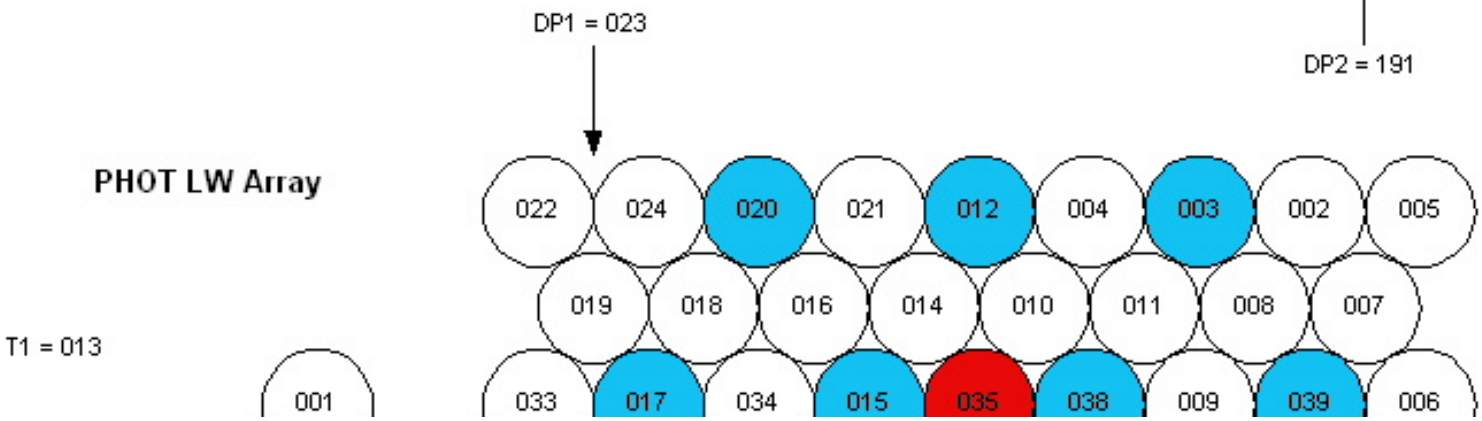


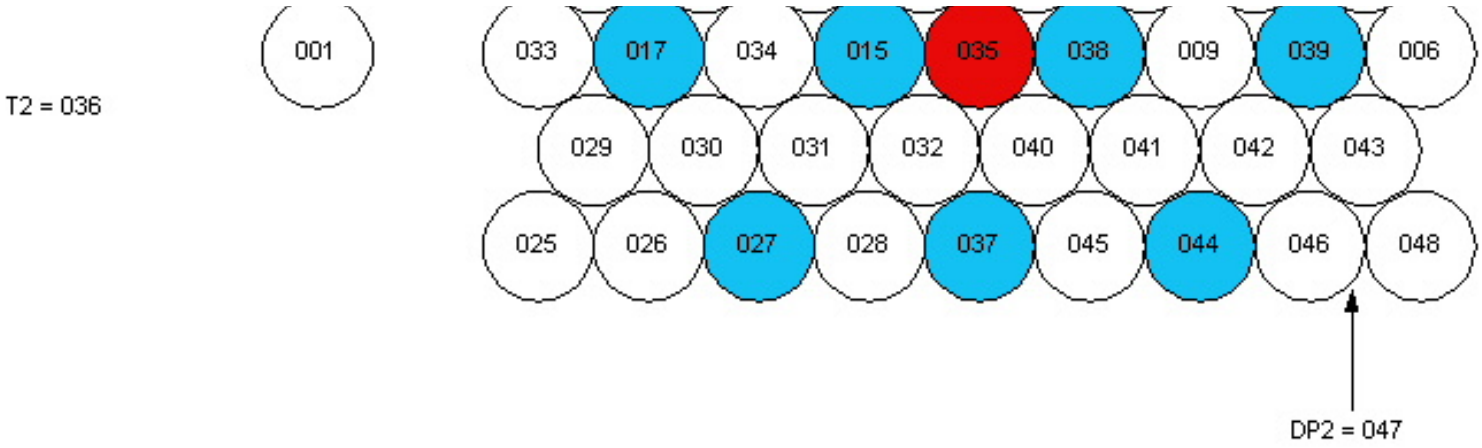

026
032
034
038
044
046
048
062
065
069
051
054
056

Long Photometer Array

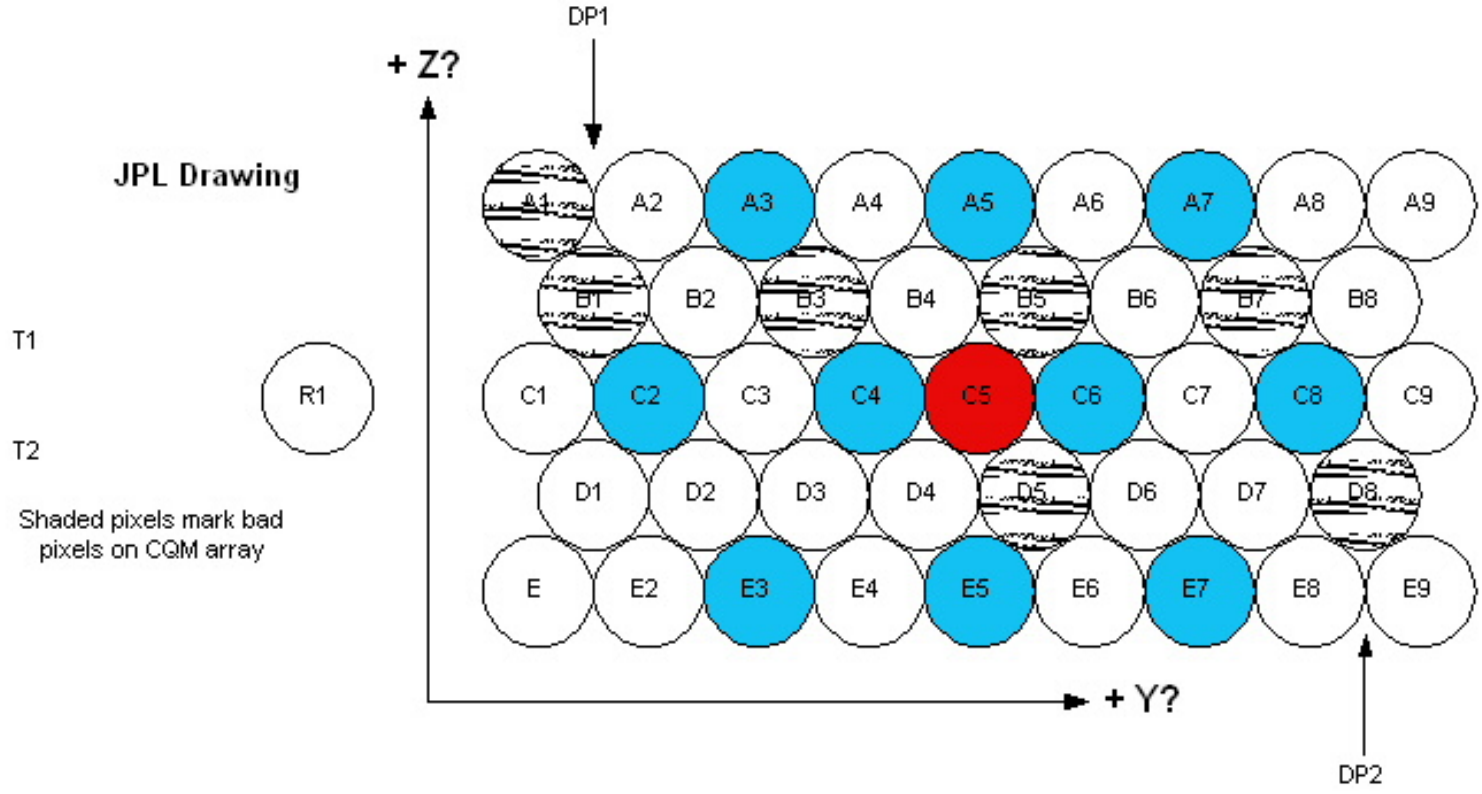






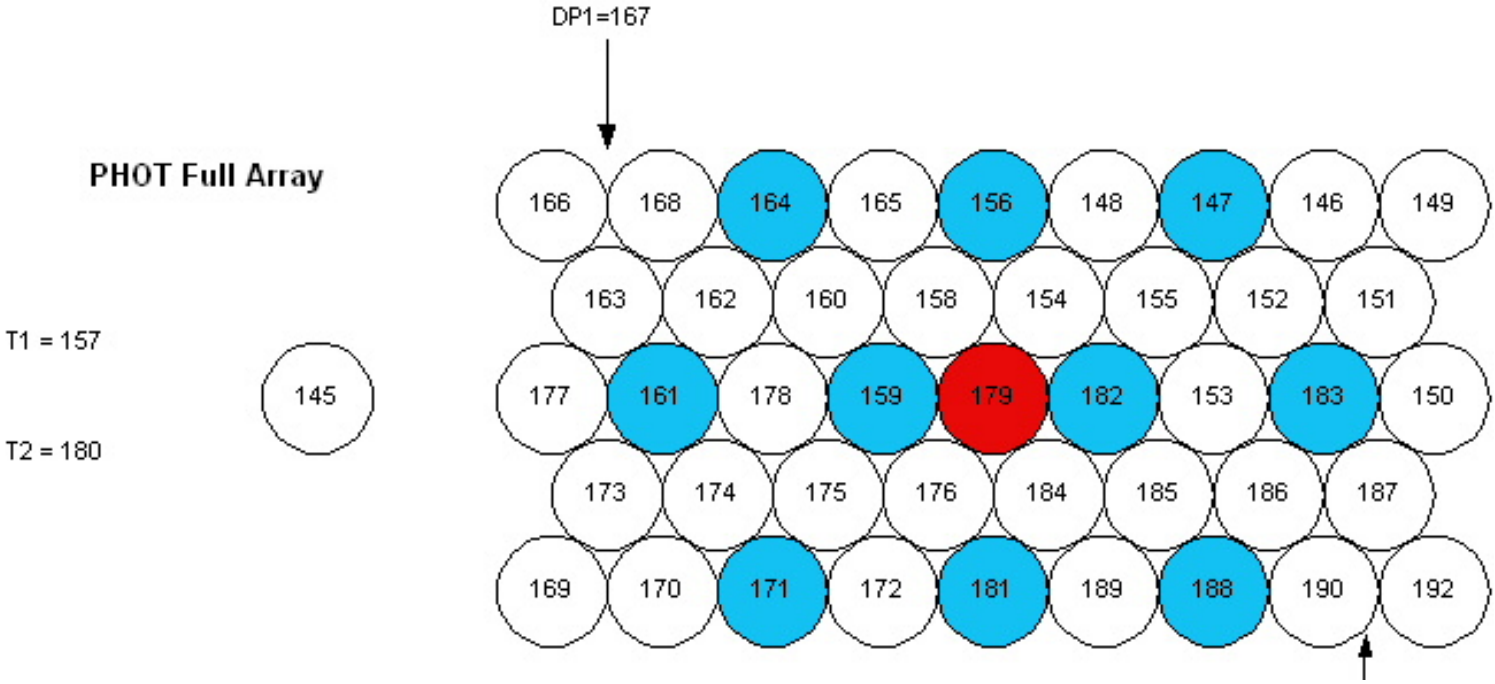


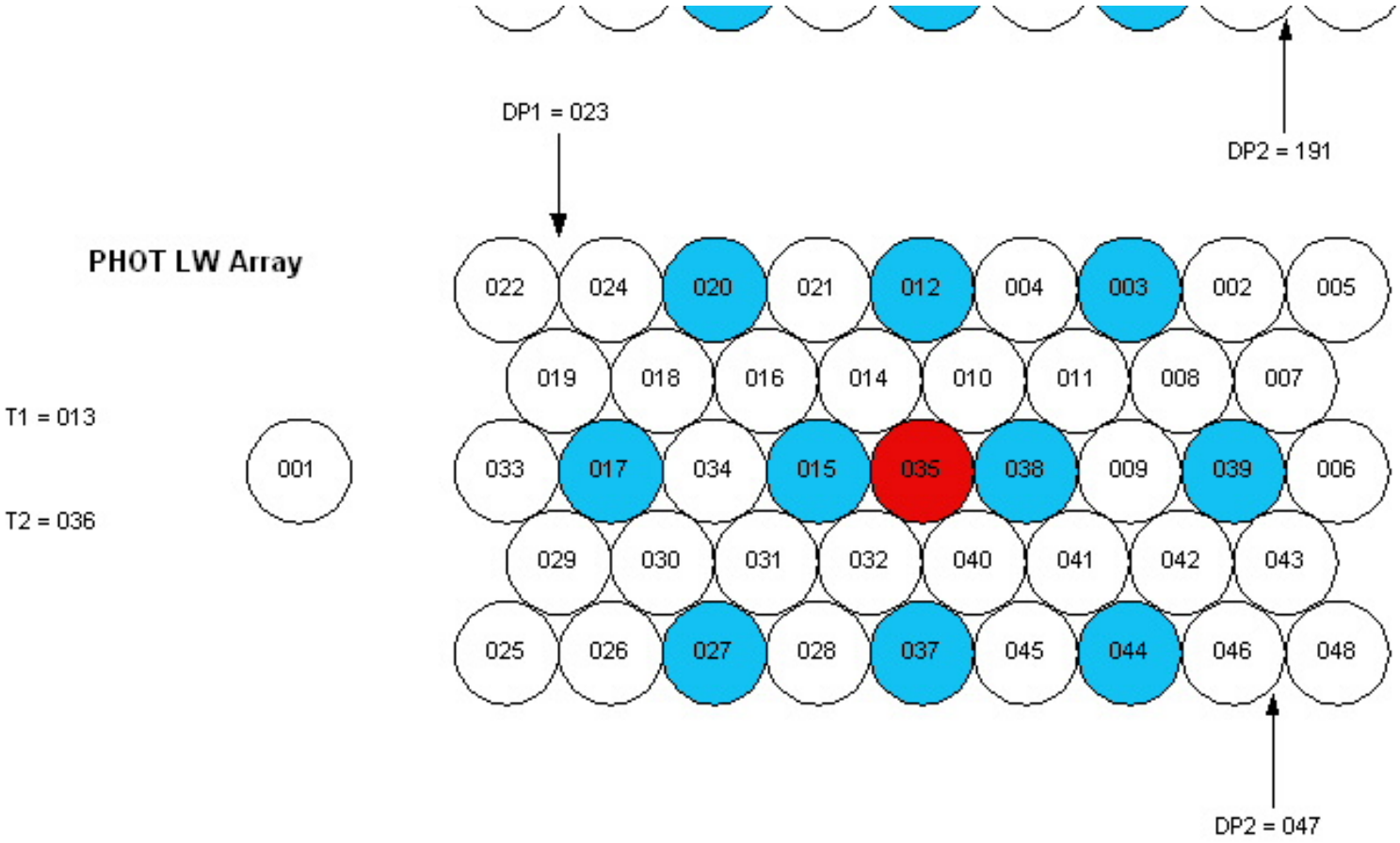
JPL Drawing



Shaded pixels mark bad pixels on CQM array

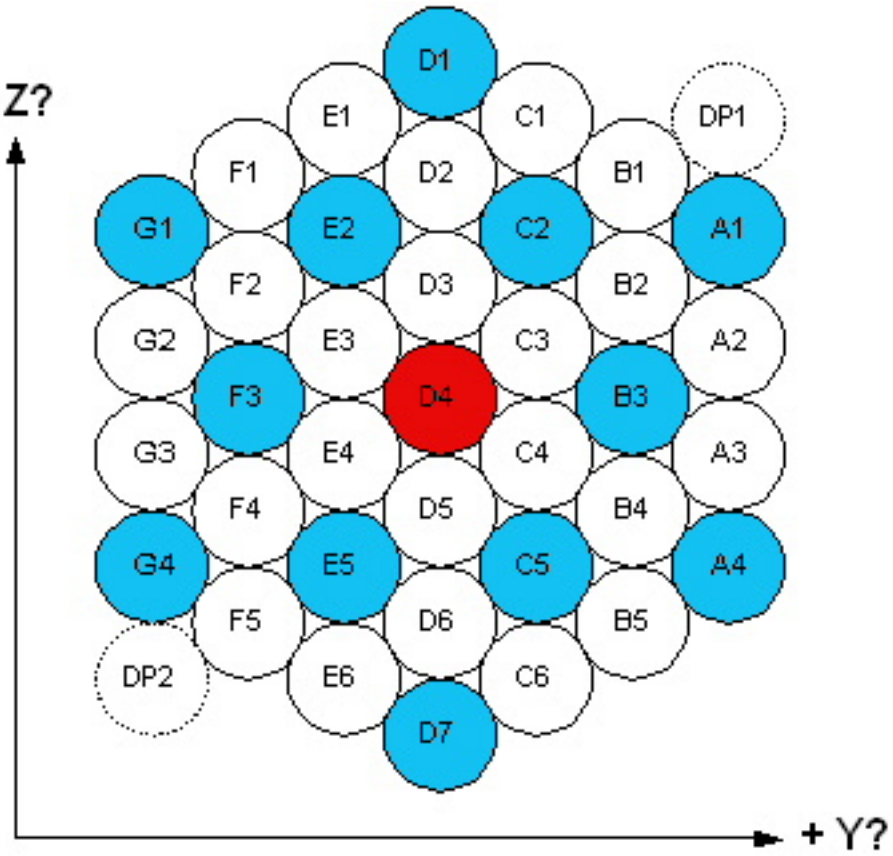
PHOT Full Array




DP1 = 023
DP2 = 191
PHOT LW Array
022 024 020 021 012 004 003 002 005
019 018 016 014 010 011 008 007
033 017 034 015 035 038 009 039 006
029 030 031 032 040 041 042 043
025 026 027 028 037 045 044 046 048
T1 = 013
001
T2 = 036
DP2 = 047

Short Spectrometer Array

JPL Drawing

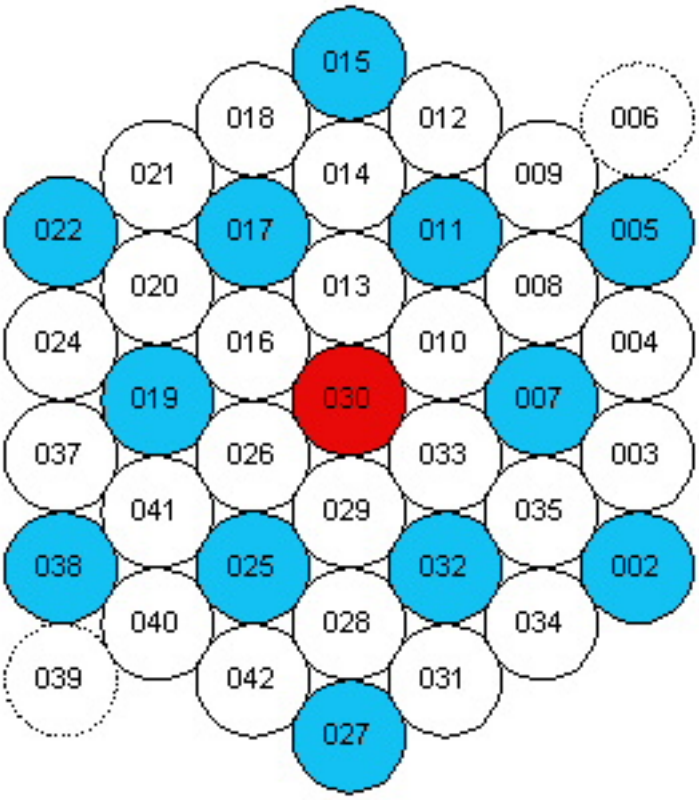


SPEC Full Array

R1 = 001

T1 = 023

T2 = 036

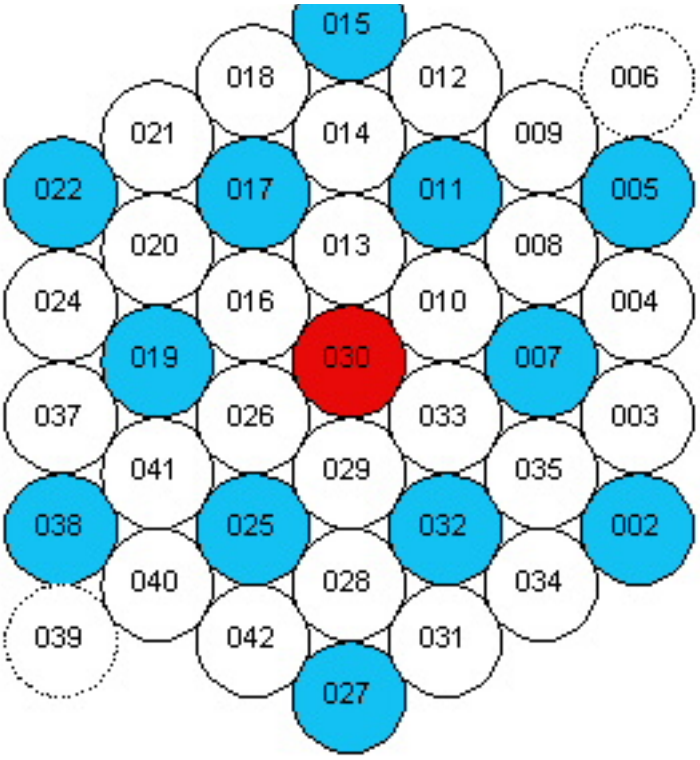


SPEC SW Array

R1 = 001

T1 = 023

T2 = 036



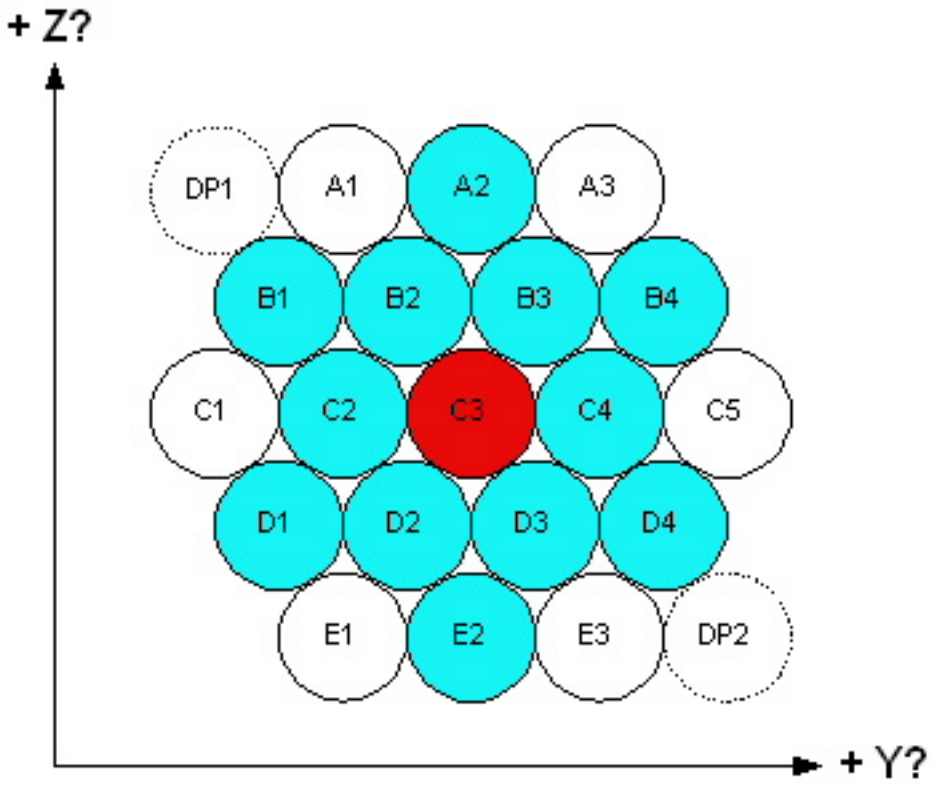
Long Spectrometer Array

JPL Drawing

R1

T1

T2

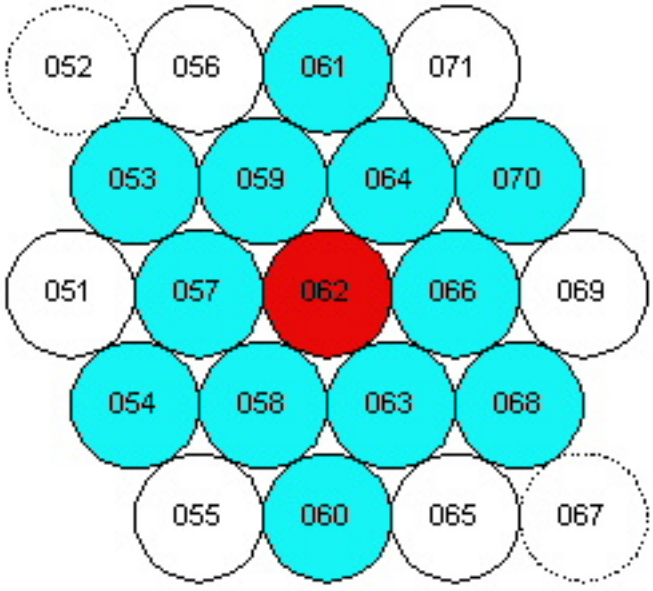


SPEC Full Array

R1 = 049

T1 = 050

T2 = 072



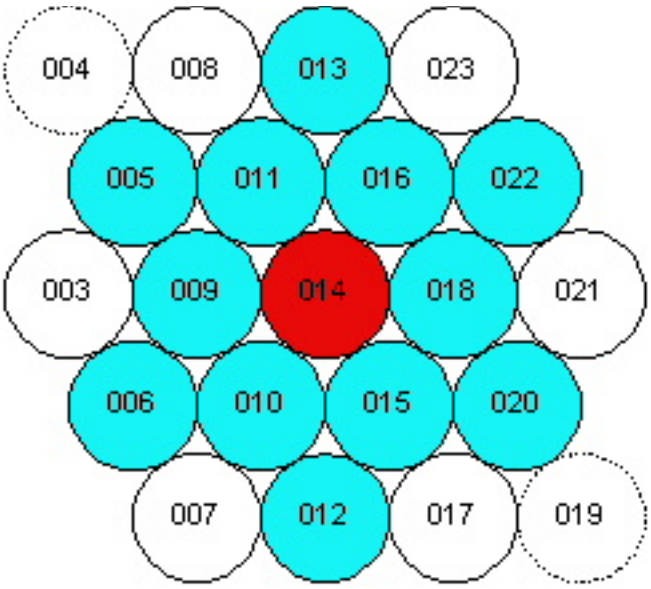
SPEC LW Array

SPEC LW Array

R1 = 001

T1 = 002

T2 = 024



A Note on the Time Format

Unless otherwise specified, times displayed in *QLA* are TAI (Temps Atomique International), measured in seconds, with epoch 1st Jan 1958.