The Legacy Remote Mode for HS Detector Control

This manual explains how to set up a client compatible with the program ***hsserver_legacy***.

Introduction

The legacy remote mode server is a method of controlling the Rayonix HS (High Speed) series of detectors by emulating the old *marccd* style of remote mode. In this way the detector can be controlled by the user's control software. Data acquisition controls, such as changing binning, collecting data images and data series, setting header information, and saving files are available through this interface. An institution might prefer to use this mode if they have previously controlled Rayonix detectors via *marccd* remote mode and want to quickly get going with minimal changes to their control software.

Legacy remote mode communication diagram

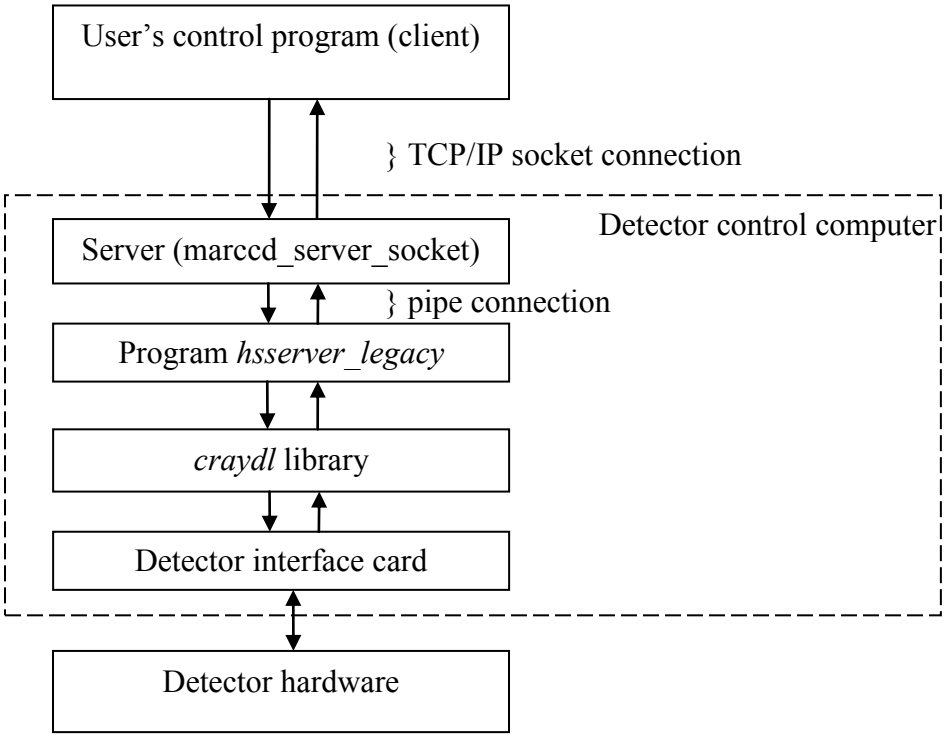

Figure 1 – Legacy remote mode communication path

Configuring the legacy remote mode

The file /opt/rayonix/configuration/craydl/RemoteModeEmulator.conf contains the following configurable variables (with suggested defaults in the provided config file).

- **ServerEnvironment:** if an environment variable is required by the control program, it can be inserted here (usually not required)
- **ServerCommand:** usually marccd_server_socket.
- **ServerArguments:** the port number to be opened by the server. It should match that looked for by the client. The sample client program provided uses port number 2002.
- **ServerLog:** not yet implemented at this time.

Typically no change would be required for these parameters.

The client program

A sample client program marccd_client_socket is included in the legacy remote mode files. It functions like a telnet session to the socket program, into which text commands (described in the next section) may be entered to drive *hsserver_legacy*. Users will need to incorporate this or a similar client into the controlling program they wish to use.

Type ./marccd_client_socket to start the program. The user may try typing in the commands below (such as get_state, or get_bin, etc.) in order to verify that indeed the *hsserver_legacy* program is executing these commands.

Alternatively, for testing purposes a telnet session may be used to connect to the server program and enter commands one at a time by hand. Typically the following command would be issued: "telnet LOCALHOST [port number]".

Remote commands

The program *hsserver_legacy* understands the following remote mode commands:

Remote Mode Command	Effect
`get_size`	*hsserver_legacy* will answer with the fast (x) and slow (y) dimensions of the data frame.
`get_size_bkg`	*hsserver_legacy* will answer with the fast (x) and slow (y) dimensions of the stored background frame (0,0 if no background frame is yet present).
`get_bin`	*hsserver_legacy* will answer with the fast (x) and slow (y) binning of the data frame.
`set_bin,x,y`	*hsserver_legacy* will set the fast (x) and slow (y) binning of the data frame.

<table>
<tr><td>start</td><td>hsserver_legacy will start integrating data (stop clearing) on the CCD.</td></tr>
<tr><td>readout,flag[,filename]</td><td>hsserver_legacy will stop integrating and start reading the CCD; given filename(s), it will queue the correction and writing of the file to disk
<table>
<tr><th>Flag</th><th>Action</th></tr>
<tr><td>0</td><td>read data into raw data frame storage</td></tr>
<tr><td>1</td><td>read data into background frame storage</td></tr>
<tr><td>2</td><td>read data into system scratch storage</td></tr>
<tr><td>3</td><td>read data into data frame storage and do NOT correct [and write uncorrected frame]</td></tr>
</table></td></tr>
<tr><td>dezinger,flag</td><td>***TO BE IMPLEMENTED*** hsserver_legacy will calculated a "dezingered" frame from two stored frames. One of the source frames is the System Scratch frame.
The second source frame and the destination are specified with the flag.
<table>
<tr><th>Flag</th><th>Action</th></tr>
<tr><td>0</td><td>use and store into the latest data frame.</td></tr>
<tr><td>1</td><td>use and store into the current background frame</td></tr>
<tr><td>2</td><td>use and store into system scratch storage (not useful; frame dezingered with itself)</td></tr>
</table></td></tr>
<tr><td>correct</td><td>hsserver_legacy will apply geometric and flatfield corrections to the raw data frame.</td></tr>
<tr><td>writefile,filename,flag</td><td>***TO BE IMPLEMENTED*** hsserver_legacy will write out a data frame to a file on disk. The parameter filename is the name of the file to be written.
<table>
<tr><th>Flag</th><th>Action</th></tr>
<tr><td>0</td><td>write raw file</td></tr>
<tr><td>1</td><td>write corrected file</td></tr>
</table></td></tr>
<tr><td>abort</td><td>hsserver_legacy will abort the current operation. Normally this would be done to stop integration and return the CCD to continuous clear mode.</td></tr>
<tr><td>get_temp</td><td>***TO BE IMPLEMENTED*** Returns the current CCD temperature, oe the highest (warmest) CCD temperature in a detector with multiple CCDs</td></tr>
<tr><td>get_press</td><td>***TO BE IMPLEMENTED*** Returns the current pressure inside the detector head</td></tr>
<tr><td>get_stability</td><td>***TO BE IMPLEMENTED*** Requires valid Baseline stabilization mode license. See manual section describing Baseline stabilization.</td></tr>
<tr><td>set_stability,target</td><td>***TO BE IMPLEMENTED*** Requires valid Baseline stabilization mode license. See manual section describing Baseline stabilization.</td></tr>
<tr><td>get_roi</td><td>***TO BE IMPLEMENTED*** Requires valid Region of Interest mode license. See manual section describing Region of Interest.</td></tr>
</table>

<table>
<tr><td>set_roi,x0,y0,x1,y1</td><td>***TO BE IMPLEMENTED*** Requires valid Region of Interest mode license. See manual section describing Region of Interest.</td></tr>
<tr><td>header,header_data\n</td><td>***TO BE IMPLEMENTED*** hsserver_legacy will accept header_data and interpret item=value pairs to be placed into the data frame header. header_data consists of a list of item=value pairs separated by commas and terminated by a newline (\n). The following items are understood:
<table>
<tr><th>Parameter</th><th>Type (Units)</th></tr>
<tr><td>detector_distance</td><td>float (mm)</td></tr>
<tr><td>beam_x</td><td>float (mm)</td></tr>
<tr><td>beam_y</td><td>float (mm)</td></tr>
<tr><td>exposure_time</td><td>float (sec)</td></tr>
<tr><td>start_phi</td><td>float (deg)</td></tr>
<tr><td>rotation_axis</td><td>string (omega, chi, kappa, phi, gamma, delta, or xtal_to_detector)</td></tr>
<tr><td>rotation_range</td><td>float (deg)</td></tr>
<tr><td>source_wavelength</td><td>float (angstroms)</td></tr>
<tr><td>file_comments</td><td>string</td></tr>
<tr><td>dataset_comments</td><td>string</td></tr>
</table></td></tr>
<tr><td>get_state</td><td>hsserver_legacy will answer with the current state of the system.

For remote mode version 1, “state” has been superseded by the more complete “status,” which is returned by the get_state command. The command get_state will return the more complex “status,” which includes the state in the lower 4 bits. Only the states IDLE, ERROR and BUSY will ever be seen. See the section below for the discussion of the version 1 protocol.

The integer numbered states possible in remote mode version 0 are:
<table>
<tr><th>State Number</th><th>State</th></tr>
<tr><td>0</td><td>IDLE</td></tr>
<tr><td>6</td><td>UNAVAILABLE</td></tr>
<tr><td>7</td><td>ERROR</td></tr>
<tr><td>8</td><td>BUSY</td></tr>
</table></td></tr>
<tr><td>set_state,state</td><td>***NOT IMPLEMENTED*** hsserver_legacy will set the state to the desired state. This is for testing purposes only and has no use in a normally functioning system.</td></tr>
</table>

<table>
<tr>
<td><code>shutter,flag</code></td>
<td>hsserver_legacy will set the shutter state to either closed or open. (Only if hsserver_legacy controls the shutter!) If the MarDTB is used, this function controls the MarDTB shutter. Otherwise, it controls the shutter attached to the shutter input (usually “Trigger” connector) on the detector head.
<table>
<tr><th>Flag</th><th>Action</th></tr>
<tr><td>0</td><td>manual/closed
Shutter signaled to close. Signal changes according to “shutter,flag” command input.</td></tr>
<tr><td>1</td><td>manual/open
Shutter signaled to open. Signal changes according to “shutter,flag” command input.</td></tr>
<tr><td>2</td><td>automatic/frame
Shutter signaled to open and close for each data frame in a series.</td></tr>
<tr><td>3</td><td>automatic/series
Shutterless data collection mode. Shutter signaled to open at the beginning of a series (either timed or triggered) and signaled to close after series is finished.</td></tr>
</table>
</td>
</tr>
<tr>
<td><code>start_series_triggered,
[,exposure_parameter]
[,n_frames]
[,first_frame_number]
[,filename_base]
[,filename_suffix]
[,number_field_width]</code></td>
<td>Starts a data collection series which is triggered by the user’s input trigger pulses (usually timed to the experimental conditions, but also could be triggered asynchronously using a pulse generator). A background image is required to have already been acquired. Exposure_parameter has the following possibilities:
<table>
<tr><th>Exposure_Parameter</th><th>Action</th></tr>
<tr><td>0 (integer)</td><td>Frame Triggered Mode. Rising edge input trigger causes frame transfer/readout and acquisition of next image.</td></tr>
<tr><td>1 (integer)</td><td>Bulb Mode. Rising edge of input trigger starts image acquisition. Falling edge causes frame transfer / readout.</td></tr>
<tr><td>T (floating point number)</td><td>Timed Triggered Mode. Floating point number (time T) is used as an exposure time. Receiving input trigger causes frame transfer / readout, and each exposure lasts time T.</td></tr>
</table>
N_frames is the number of frames in the sequence (default 1). First_frame_number is the number of the first frame for the filename (default 1). Filenames are defined as [filename base][number field][filename suffix] where number_field_width is an integer that defines the number of digits of the field.</td>
</tr>
</table>

<table>
<tr><td>start_series_timed
[,n_frames]
[,first_frame_number]
[,integration_time]
[,interval_time]
[,filename_base]
[,filename_suffix]
[,number_field_width]</td><td>Starts a data collection series which is timed by the detector's internal clock (asynchronous to the user's experiment). A background image is required to have already been acquired. N_frames is the number of frames in the sequence (default 1). First_frame_number is the number of the first frame for the filename (default 1). Integration_time is the time duration of each exposure. Interval_time is the time between the starting of each exposure (must be equal to or greater than integration_time). Filenames are defined as [filename base][number_field][filename suffix] where number_field_width is an integer that defines the number of digits of the field.</td></tr>
<tr><td>set_readout_mode,flag</td><td>Program will set the readout mode to one of the following values:
<table>
<tr><th>Flag</th><th>Readout Mode</th><th>Gain</th><th>Speed</th><th>Read bits</th></tr>
<tr><td>0</td><td>Standard</td><td>norm</td><td>norm</td><td>16</td></tr>
<tr><td>1</td><td>High Gain</td><td>high</td><td>norm</td><td>16</td></tr>
<tr><td>2</td><td>Low Noise</td><td>high</td><td>med</td><td>16</td></tr>
<tr><td>3</td><td>HDR</td><td>high</td><td>low</td><td>18</td></tr>
</table>
Note that some data processing programs may need to be updated to read 18 bit HDR (High Dynamic Range) mode files.</td></tr>
<tr><td>get_readout_mode</td><td>Returns the current readout mode setting, with flags defined above in set_readout_mode section.</td></tr>
<tr><td>set_gating,flag</td><td>During a triggered data series collection, this option uses the 2nd input trigger as a gate to make the detector insensitive to framing triggers on the 1st input trigger.
<table>
<tr><th>Flag</th><th>Action</th></tr>
<tr><td>0</td><td>Not gated. Triggers from input trigger 1 function normally.</td></tr>
<tr><td>1</td><td>Gated. When input trigger two is in TTL high state, framing triggers from input trigger one will be ignored.</td></tr>
</table></td></tr>
<tr><td>get_gating</td><td>The current gating setting (integer) is returned.</td></tr>
<tr><td>end_automation</td><td>hsserver_legacy will exit remote mode.</td></tr>
</table>

Note on command handling by server application

In addition to the above commands, it is recommended that any server application implements the following commands (already implemented in the provided marccd_server_socket:

Command to server program	Action
get_state	same as above, but queries from the client should be answered directly by the server without querying *hsserver_legacy*.
get_size	same as above, but queries from the client should be answered directly by the server without querying *hsserver_legacy*.
get_size_bkg	same as above, but queries from the client should be answered directly by the server without querying *hsserver_legacy*.

`get_frameshift`	same as above, but queries from the client should be answered directly by the server without querying *hsserver_legacy*.
`get_bin`	same as above, but queries from the client should be answered directly by the server without querying *hsserver_legacy*.
`get_state_hist`	(Implemented completely in the server.) Anwers with the current state and the most recent previous state, separated by commas. (See `get state`.)

State and status values in remote mode version 1

In the version 1 protocol, the status of each task is represented in a 4 bit field in the 32 bit state value. To use version 1 instead of version 0, include the appropriate configuration file, marccd_server_v1.conf, instead of the older marccd_server.conf file. This file contains the parameter "`remote_mode_version`" set to 1.

The task values are:

Task Number	**Task**
0	TASK_ACQUIRE
1	TASK_READ
2	TASK_CORRECT
3	TASK_WRITE
4	TASK_DEZINGER
5	TASK_SERIES

The status bits for each task are:

Task Status Bit	**Task Status**
0x1	TASK_STATUS_QUEUED
0x2	TASK_STATUS_EXECUTING
0x4	TASK_STATUS_ERROR
0x8	TASK_STATUS_RESERVED

Therefore, the state value looks like Figure 2, with eight four-bit fields

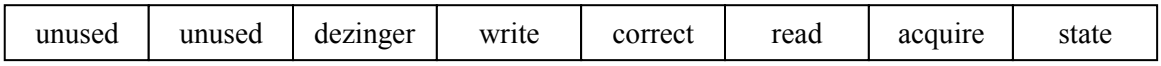

Figure 2 - State fields in remote mode version 1

Examples state values returned by get_state:

Idle	0x00000000
Busy (interpreting command)	0x00000008
Error (command not understood)	0x00000007
Acquiring	0x00000010

Reading	0x00000200
Reading w/correct and write queued	0x00011200
Correcting w/write queued:	0x00012000
Error writing file	0x00040000

These are the C definitions of masks for looking at task state bits:

```
#define STATUS_MASK             0xf

#define TASK_STATUS_MASK(task)  (STATUS_MASK <<
(4*((task)+1)))
```

These are some convenient macros for checking and setting the state of each task. They are used in the *hsserver_legacy* code and can be used in the client code:

```
#define TASK_STATUS(current_status, task) (((current_status)
& TASK_STATUS_MASK(task)) >> (4*((task) + 1)))

#define TEST_TASK_STATUS(current_status, task, status)
(TASK_STATUS(current_status, task) & (status))
```

The following is an example of pseudo C code to do an exposure sequence:

```
/* Get a backround frame */
   /* Wait for detector to NOT be reading */
   do {
     /* send: get_state */
     /* put result in state */
   }  while (TEST_TASK_STATUS(state, TASK_READ,
TASK_STATUS_EXECUTING));

   /* send: readout,1 */

/* Get a 2nd backround frame - This (readout; dezinger) can be
repeated if desired */
   /* Wait for detector to NOT be reading */
   do {
     /* send: get_state */
     /* put result in state */
   }  while (TEST_TASK_STATUS(state, TASK_READ,
TASK_STATUS_EXECUTING));

   /* send: readout,2 */

/* Dezinger to combine 2 background frames into low noise dezingered
 * background frame */
   /* Wait for detector to NOT be reading */
   do {
     /* send: get_state */
```

```
     /* put result in state */
   }  while (TEST_TASK_STATUS(state, TASK_READ,
TASK_STATUS_EXECUTING));

   /* send: dezinger,1 */

/* Get a sequence of data frames */
     while(1) {

        /* Wait for detector to NOT be acquiring (i.e. it has at least
         * started the previous read) */
           do {
              /* send: get_state */
              /* put result in state */
           }  while (TEST_TASK_STATUS(state, TASK_ACQUIRE,
TASK_STATUS_EXECUTING));

        /* Start detector frame acquisition */
           /* send: start */

        /* Wait for detector to start acquiring (this is very
         * important, so that no X-rays are on the detector during
         * readout; here could be a delay of approximately the
         * readout time) */
           do {
              /* send: get_state */
              /* put result in state */
           }  while (!TEST_TASK_STATUS(state, TASK_ACQUIRE,
TASK_STATUS_EXECUTING));


        /* Do exposure "stuff" here */


        /* End acquisition by starting readout, (correction and write
         * will be automatically queued and executed.) */

           /* send: readout,0,filename */
     }
```

Information on background frames and some sample data collection routines

The following are possible sequences of commands that you may implement in your remote mode control of *hsserver_legacy*. We assume here that your facility has implemented its own shutter control.

Either a “bias” frame (a background with zero integration time) or a non-zero time “dark” frame must always be collected and put in the Background buffer, to be subtracted from the data. Because of the extremely low CCD operating temperature, our X-ray detectors have minimal

dark current; thus taking the time to collect a dark frame (as opposed to a bias frame) is usually not necessary, even for very long x-ray exposure times of data.

Here is the simplest and quickest method of collecting a Background image (not recommended):

- [CLOSE SHUTTER] *(make sure shutter is closed)*
- `start` *(start integration)*
- `readout,1` *(read data into both raw and background buffers)*

The reason it is not recommended is that this method will potentially have zingers in the image. Zingers in the background will be subtracted from data images, leaving the final images with zero intensity spots. In addition, one background can be used multiple times and therefore a zinger in a background will contaminate several images.

Here is a sequence that will make a dezingered bias frame (recommended method):

- [CLOSE SHUTTER] *(make sure shutter is closed)*
- `start` *(start integration)*
- `readout,2` *(read and copy to Scratch buffer)*
- `start`
- `readout,1` *(read and copy to Background buffer)*
- `dezinger,1` *(dezinger from Background and Scratch data, put image in Background buffer)*

The background doesn't have to be retaken for every data image taken, but generally should be retaken at the start of every new data set, or once every half hour, whichever is sooner (depending on the thermal stability of the hutch). For the SX Series detector, if a mismatch in the level of the 4 quadrants of data frames is noticed, the bias is probably drifting and should be recollected (and maybe should be set to be collected more often).

To collect a data image:

- `start` *(start integration)*
- [OPEN SHUTTER]
- [WAIT DESIRED TIME]
- [CLOSE SHUTTER]
- `readout,0,FILENAME` *(read data into raw frame buffer; queue the correction; corrected data are written to the filename)*

Note that in normal operation, neither the background frame nor the raw (uncorrected) data frame need to be saved.

Here is a sequence of commands for taking a dezingered data frame:

- `start` *(start first integration)*
- [OPEN SHUTTER]
- [WAIT TIME1]
- [CLOSE SHUTTER]
- `readout,2` *(read data into raw buffer and copy to Scratch)*
- `start` *(start second integration)*
- [OPEN SHUTTER]
- [WAIT TIME2]
- [CLOSE SHUTTER]
- `readout,0` *(read data into raw frame buffer)*
- `dezinger,0` *(dezinger from raw and Scratch data; data sent to raw buffer)*
- `correct` *(apply correction; data sent to "corrected" buffer)*
- `writefile,IMAGE,1` *(write data from corrected frame buffer to file)*

The dezinger operation goes through every pixel of the two (or multiple) separate reads of the detector, and compares the values. If the two values are very different, as determined by a statistical test, then the lower value is accepted and the higher value is discarded. If the values are statistically close enough, then they are averaged.

Because a statistical test is used, special care must be taken to make dezingered data frames. Each exposure must truly be the same (same X-ray dose, same movement of the sample or no movement of the sample, and very little decay or other change in sample). Otherwise the dezinger operation will yield unpredictable results.

If the source has constant intensity, then TIME1 = TIME2 = total_time/2. However, if the source has a short decay time, then the times must be TIME2>TIME1, calculated so that that both frames have equal dose, within a few percent.

Compiling the sample programs

The source programs to run hsserver_legacy are typically located in /opt/rayonix/src/marccd_server. Along with these instructions you should obtain a tar file called example_remote_server.tgz). If you have not already done so, unzip and untar the file in a new directory by typing "`tar -zxvf example_remote_server.tgz`."

Included in the untarred files will be:

dsmar_utils.c
dsmar_utils.h
Makefile
Makefile.bak
marccd.c
marccd_client_socket.c

marccd_server_pipe.c
marccd_server_socket.c
remote_mode_manual.pdf
socket_utils.c
socket_utils.h

Before compiling any programs, type "`make depend`" in the current directory to update the dependencies in the Makefile to match the compiler libraries on your computer.

Compile marccd_client_socket.c and marccd_server_socket.c by typing "`make marccd_client_socket`" and "`make marccd_server_socket`." The file marccd_server_pipe.c is also provided as a sample to show how a connection can be made with *hsserver_legacy* using pipes, but in the example that follows, the programs with socket connections are used.

Compliled versions of marccd_server_socket and marccd_client_socket should be installed in /opt/rayonix/bin.