The JCMT Observing Queue and Recipe Sequencer

Russell D. Kackley*, Nicholas P. Rees, Craig Walther, Tim Jenness
Joint Astronomy Centre, 660 N. ʻA’ohoku Place, Hilo, HI, USA 96720

ABSTRACT

The James Clerk Maxwell Telescope (JCMT), the world's largest sub-mm telescope, will soon be switching operations from a VAX/VMS based control system to a new, Linux-based, Observatory Control System (OCS). A critical part of the OCS is the set of tasks that are associated with the observation queue and the observing recipe sequencer: 1) the JCMT observation queue task, 2) the JCMT instrument task, 3) the JCMT Observation Sequencer (JOS), and 4) the OCS console task. The JCMT observation queue task serves as a staging area for observations that have been translated from the observer's science program into a form suitable for the various OCS subsystems. The queue task operates by sending the observation at the head of the queue to the JCMT instrument task and then waits for the astronomer to accept the data before removing the observation from the queue. The JCMT instrument task is responsible for running up the set of tasks required to observe with a particular instrument at the JCMT and passing the observation on to the JOS. The JOS is responsible for executing the observing recipe, pausing/continuing the recipe when commanded, and prematurely ending or aborting the observation when commanded. The OCS console task provides the user with a GUI window with which they can control and monitor the observation queue and the observation itself. This paper shows where the observation queue and recipe sequencer fit into the JCMT OCS, presents the design decisions that resulted in the tasks being structured as they are, describes the external interfaces of the four tasks, and details the interaction between the tasks.

Keywords: Instrument control, observatory control, user interface, sequencer, observing recipe, XML, DRAMA

1. INTRODUCTION

The JCMT has operated its instruments for many years using software written for the VAX/VMS operating system. With the impending arrival of a new front-end receiver, HARP-B\textsuperscript{2} (Heterodyne Array Receiver Program – B band), and a new back-end data acquisition system, ACSIS\textsuperscript{3} (Auto-Correlation Spectral Imaging System), the JAC, operator of the JCMT, is developing a new, Linux-based, Observatory Control System (OCS) to support this new hardware. Some of the existing VAX/VMS software will still be maintained to support legacy receivers, but most of the software directly used by observers and telescope operators will be replaced. Commissioning of the new OCS is expected to take place in late 2004. This paper describes the software associated with the observing queue and recipe sequencer.

2. SYSTEM DESCRIPTION

Figure 1 provides an overview of the tasks involved in a typical observation at the JCMT. The primary focus of this paper is the tasks enclosed in the area labeled Observatory Control System. An astronomer uses the observation preparation tool to specify the details of their observations (usually in advance of their observing run) and store them in the observation database. Figure 2 provides a chart showing the flow of a typical observing run when the astronomer is at the JCMT. The JCMT OCS makes use of a concept known as the Minimum Schedulable Block (MSB), which is a set of observations that must be performed in its entirety in order to produce a complete, fully reducible set of data. The astronomer uses the observation query and selection tool to query the database in order to select a set of MSB’s appropriate to the current observing conditions (e.g., weather and local sidereal time). The astronomer selects the MSB’s to be executed and sends them to the translator so that their component observations can be translated into a form suitable for the various telescope and instrument subsystems. The translator also places the observations on the observing queue.

* For further author information, contact Russell Kackley (r.kackley@jach.hawaii.edu)
Figure 1. JCMT Observing Tasks

Observation database

Observation query and selection

Observation translation

Observation queue

Observation recipe sequencer

OCS console (GUI)

JCMT instrument task

Telescope control sub-systems

Instrument control sub-systems

Data acquisition sub-systems

Observation Management System

Observation Control System
Figure 2. JCMT Observing Flow

1. Query observation database
2. Select MSB(s)
3. Translate MSB(s)
4. Enqueue observation(s) from MSB(s)
5. Queued observations?
   - Y: Send next queued observation to JCMT instrument task
   - N: Stop queue
6. Successful completion?
   - Y: Mark observation as “Observed”
   - N: Run up necessary tasks
7. Execute observing recipe

---

720 Proc. of SPIE Vol. 5496
When the astronomer starts the queue, the observation at the head of the queue is sent to the JCMT instrument task. The instrument task reads the observation description, determines which instrument is being used for the observation, runs up the necessary tasks for that instrument, and sends the observation to the observation recipe sequencer. The sequencer reads the observation description, determines the observing recipe name and parameters, and sends commands to the telescope and instrument subsystems instructing them on how to perform the observation. When an observation is complete (or if it fails prior to completion), the recipe sequencer reports the completion status (success or failure) to the queue task. If the observation has completed successfully, the queue task sends the next observation to the sequencer and the cycle described above is repeated for the new observation. If the observation failed to complete, or completed unsuccessfully, the queue task marks the observation as failed, stops processing observations, and waits for the astronomer to decide how best to proceed. When all the observations for an MSB have completed, the queue task prompts the astronomer to either accept or reject the data. If the astronomer accepts the data, the observation database is informed that the MSB has been completed and the observations for that MSB are removed from the queue. If the astronomer rejects the data, the observations for the MSB remain on the queue and the astronomer can repeat the observations or delete them from the queue. During all this activity, the OCS console task, which provides a graphical user-interface to the astronomer, is monitoring the progress of the observation and informing the astronomer of the state of the observation and the various subsystems.

The following sections provide more detail on the queue, JCMT instrument, recipe sequencer, and user-interface tasks. The common theme that drove the most important design decisions for these tasks was to keep their external interfaces as simple as possible in order to keep the commands flowing from one task to the next at a relatively high level and let the tasks themselves handle complex issues internally.

### 3. OBSERVING QUEUE

The observing queue manages the list of observations that the astronomer has selected to be executed and makes sure that the observations are sent to the telescope in the correct order and as efficiently as possible. The design of the queue was greatly influenced by the original JCMT observing queue used with UKT14 and SCUBA but with additional requirements that the queue should be able to queue observations for multiple instruments, it should be portable enough to run on UKIRT as well as JCMT without modification, and should be intelligent enough to realize when runtime decisions need to be made (either by the software or observer) prior to sending an observation to the telescope.

One of the key differences between the early queues and the current OCS queue is that this queue is not implemented as a stack with observations slowly moving to the top of the queue, being observed and then being deleted from the queue but instead uses a highlight position to indicate the current observation and the next observation. As observations are taken the highlight moves down the observation list. This is a necessary change since it increases observing flexibility by allowing an observer to re-send an observation without having to re-fetch and re-translate the observation from the database. This is sometimes important since the automated reduction pipeline will not be able to indicate the signal-to-noise or data quality until after the next observation has begun. Entries are only cleared from the queue when the MSB retrieved from the database is completed and accepted by the observer.

In the sub-mm regime, the choice of pointing source and flux calibrator critically depends on the time of observation and so cannot always be planned in advance by the principal investigator when they prepare their science program. The queue overcomes this problem by recognizing that certain observations in the queue do not include targets and when they are about to be observed a short list of recommended targets is presented to the observer (as a list and also as a graphical display of the sky). For pointing observations, this list is sorted by distance from the subsequent science observation (the queue scans forward to determine which observation is relevant).

In some cases, mainly associated with legacy observing with the old VAX-based SCUBA system, the queue must optimize time-critical observation parameters before sending the observation to the telescope. For SCUBA the most common decisions are to optimize the scan angle for non-square map areas (since SCUBA is on the Nasmyth platform without an image rotator and also since there are only three optimal scan angles that result in a fully-sampled scan, the choice of optimal scan angle depends on the time of observation) and to optimize polarimeter angles for maximum angle coverage (the polarimeter angles can only be specified in Nasmyth coordinates).
The queue itself is implemented as a DRAMA \cite{7,8} task to conform to standard JCMT practice. As expected, low-level commands are available, including loading observations on to the queue (at the front, back, or at the position of the current highlight), clearing the queue, replacing the target coordinates, and stopping and starting the queue. The queue state is retained in DRAMA parameters that can be monitored by the queue monitor (described later in this paper). This approach, of separating the queue from the control GUI, allows multiple GUIs (e.g. the observer, telescope operator and support scientist in Hilo) to view and control the queue rather than restricting access to a single user.

Internally, the queue is designed to be flexible enough to allow different instruments to be queued and even to allow different backend systems to be contacted (this is important since the queue has had to work with the legacy SCUBA system and UKIRT before being released with the JCMT OCS upgrade). This is achieved using object-oriented techniques where, for example, each entry in the queue is represented by an object providing a standard interface with instrument specific differences being implemented as sub-classes. Furthermore, the DRAMA part of the queue is restricted to a single module that translates DRAMA requests to methods on the internal queue object. This approach will allow us to reuse the same queue classes in a non-DRAMA environment with minimal changes.

The queue is written in object-oriented Perl (the previous queue was written in Starlink ADAM Fortran), allowing for extremely fast development and debugging cycles and also allowing re-use of code from our other systems (especially the Observation Management Project implementation\cite{4}).

4. JCMT INSTRUMENT TASK

The JCMT Instrument Task (JCMT\_INST) is responsible for running up all tasks that are needed for a particular instrument and relaying the observation instructions to the observation recipe sequencer. Similar to many of the other tasks in the OCS, JCMT\_INST is implemented as a DRAMA task, written in Perl. Much of its design was based on a similar task, UKIRT\_INST, which runs at UKIRT as part of the UKIRT OCS.

JCMT\_INST reads an initialization file that tells it which tasks are needed for each instrument. It has actions that can be invoked to run up/down the tasks for a particular instrument. During normal observing, tasks are run up/down as a group of tasks that are associated with a particular instrument. At JCMT, several of the back-end tasks are used by more than one of the front-end instrument tasks. When a task is run up by JCMT\_INST, it keeps track of which instruments need to use that task. When a particular instrument is run down, tasks on which other running instruments are dependent are not run down. This means that tasks shared by more than one instrument will not be run down until the last instrument needing that task is run down. Another decision that is made by JCMT\_INST is to determine from the observation description whether the observation is intended for SCUBA, the JCMT’s bolometer array (whose software runs on a legacy VAX/VMS system), or for one of the JCMT heterodyne receivers (which have software interfaces running on Linux computers). The reason that JCMT\_INST must make this decision is that SCUBA observations do not make use of the OCS recipe sequencer task, so JCMT\_INST must communicate with the legacy SCUBA software, rather than the OCS software that is the subject of this paper.

JCMT\_INST also maintains a set of DRAMA parameters that can be monitored by other tasks, such as the currently running front-end receiver, and lists of the instruments and tasks that have been run up.

Finally, JCMT\_INST provides the capability to run individual tasks up or down as needed during engineering tests. This facility makes it possible to fix a bug in one task and run that task down/up without having to run down the whole system.

5. JCMT OBSERVATION RECIPE SEQUENCER

The JCMT Observation recipe Sequencer (JOS) manages the execution of the observation by sending commands to the telescope and instrument subsystems in accordance with the specified recipe. The JOS is implemented as a DRAMA task, written in Perl.
One important aspect of the OCS design is that all observations can be described in terms of one of the following six recipes:

- **Pointing** – Determine the telescope pointing offsets
- **Focus** – Determine the optimal position of the secondary mirror
- **Grid-position switch** – Take data at a set of discrete grid points
- **Raster-position switch** – Take data while scanning an area of the sky
- **Jiggle-chop** – Sample an area of the sky by taking data at a set of jiggle positions
- **Jiggle-frequency switch** – Sample an area of the sky at different frequencies by taking data at a set of jiggle positions and switching the front-end receiver tuning parameters

A second important aspect of the OCS design is that the interface between the JOS and the telescope and instrument subsystems is very simple, consisting of the following four actions:

- **CONFIGURE** – Instructs the task to read the specified XML file and configure itself according to the contents of that file.
- **SETUP_SEQUENCE** – Instructs the task to prepare itself for the next integration sequence. A small set of parameters is sent to the task along with this action to specify the unique properties of the upcoming integration sequence. The sub-systems use the supplied parameters to complete their configuration. They can return to the JOS a set of parameters that place constraints on the number of steps to be executed on the next integration sequence.
- **SEQUENCE** – Sends to each task the start and end sequence numbers for the integration sequence. The Real Time Sequencer manages the state transitions between steps of the sequence via hardware control lines.
- **END_OBSERVATION** – Instructs the tasks to end the observation.

Figure 3 shows an example recipe – the raster-position switch recipe. This recipe works by performing a calibration before the first scan row and then iterating through the scan rows until the specified area of the sky has been covered. The astronomer can specify that the system should do additional calibrations every ROWS_PER_REF number of rows.

Each recipe makes use of primitive functions. References to these primitives are shown in boldface type in Figure 3. Each primitive works by sending, in parallel, a command to each of the participating sub-systems and waiting for all sub-systems to report back that they have finished their assigned action. The most important primitives are `configure`, `calibrate`, and `integrate`. The `configure` primitive sends to each sub-system the name of the configuration file that describes the next observation. The `calibrate` primitive works by instructing the sub-systems to sample an internal calibration source, then move the telescope to a blank part of the sky and sample the sky, and then sample a second internal calibration source. The `calibrate` primitive makes use of the `integrate` primitive to perform the data-taking operations. The `integrate` primitive works by first sending information to all the sub-systems to prepare them for the next portion of the observation, called an integration sequence. When all the sub-systems report to the JOS that they are prepared, the `integrate` primitive sends to all the sub-systems the command to start the integration sequence. When all sub-systems report that they have completed the integration sequence, the `integrate` primitive returns back to the main recipe code. If any sub-system reports an error, the JOS informs all the other sub-systems that an error has occurred and stops execution of the recipe.

Another important primitive is the `breakPoint` primitive. This primitive is used at carefully selected points in the recipes. Its purpose is to provide a point at which the observation can be ended and the data gathered up to that point would have sufficient calibrations so that the data reduction process will produce a sensible data output file. The JOS provides the capability to either pause or end the observation at a `breakPoint`.
# Main observing routine for a raster-pssw observation.
sub raster_pssw( $$% ) {
  my ( $tasks,        # Reference to an array of participating tasks -
    $configuration, # Name of configuration XML to use
    $par) = @_;    # Hash containing the recipe parameters. The hash
    # must contain values for the following keys, except for
    # REFS_PER_CAL and N_CALSAMPLES, which are optional:
    # NUM_CYCLES      => Number of times to repeat entire recipe
    # STEP_TIME       => RTS step time during an RTS sequence
    # ROWS_PER_REF    => Number of rows per reference
    # REFS_PER_CAL    => Number of references per complete calibration
    # N_CALSAMPLES    => Number of samples per load on cal.

  my $status = 1;
  configure( $tasks, $configuration, $status );
  my $i = 0;
  for ( $i = 1; $i <= $par{NUM_CYCLES} && $status; $i++ ) {
    # Update the for loop parameters so the ocs_console can display current loop values
    recipeForLoopUpdate( 'CYCLE', $i, $par{NUM_CYCLES}, $status );
    my $row = 0;
    my $refs = 0;
    my $nsteps = 1;
    my $nstepsSinceLastRef = 0;
    while ($nsteps) {
      $row++;
      if (((($row - 1) % $par{ROWS_PER_REF}) == 0) {
        if ($par{REFS_PER_CAL} > 0 && ($refs % $par{REFS_PER_CAL}) == 0) {
          print "Calibrating before point $row \n";
          calibrate( $tasks, $par{STEP_TIME}, $par{N_CALSAMPLES}, $par{N_CALSAMPLES},
                      $par{N_CALSAMPLES}, "OFFSETZERO", $row, 0, $status );
        } else {
          print "Taking a reference before $row \n";
          my $nRefSamples = getNRefSamples($nstepsSinceLastRef, $par{ROWS_PER_REF}, $row,
                                $par{STEP_TIME}, $status);
          calibrate( $tasks, $par{STEP_TIME}, 0, $nRefSamples, 0, "OFFSETZERO", $row, 0,
                                $status );
        $nstepsSinceLastRef = 0;
        } $refs++;
      } else {
      # Set up the observe area parameters so the ocs_console can display them
      activateObsArea();
      print "Integrating on $row \n";
      $nsteps = integrate( $tasks,
        "SOURCE" => "SCIENCE", "INDEX" => $row, "BEAM" => "MIDDLE",
        "STEP_TIME" => $par{STEP_TIME}, "FE_STATE" => "OFFSETZERO",
        "GROUP" => $row, "DRCONTROL" => 0, "LOAD" => "SKY",
        "MASTER" => "", 1, 1, $status);
        # Keep a running total of the number of steps since the last reference
        $nstepsSinceLastRef += $nsteps if ($status);
      } deactivateObsArea(); # Remove the observe area parameters
      # Provide a breakPoint in case we want to end the observation at the end of a CYCLE
      breakPoint( $status );
    }
  recipeForLoopRemove( 'CYCLE', $status ); # Remove the for loop parameters
  end_observation( $tasks );
}

In a perfect world, every observation started by an astronomer would run successfully to completion with no human intervention required. However, in the real world it is frequently necessary to pause midway through an observation and evaluate the data gathered so far to see if the sub-systems are operating correctly. In the worst case, it may even be necessary to abort an observation before it has completed because of a failure in some part of the observing system. The JOS provides DRAMA actions that allow the astronomer to do the following:

- Pause/continue an observation at any time
- Pause at a breakpoint/undo pause at a breakpoint
- End the observation at the next breakpoint and write data files
- End the observation after the current sequence and write data files
- Abort the observation immediately (do not write data files)

Another important function of the JOS is to manage the telescope beam control. For efficient operations, the JCMT requires that we be able to run noise references on SCUBA while observing with HARP-B, or, observe with SCUBA while running engineering tests with HARP-B. Therefore, the JOS implements two actions, OBSERVE and LOAD, that can be used to perform an observation. The OBSERVE action is used when a science observation is being run, in which case the JOS will send commands to the Telescope Control System (TCS) to move the telescope. On the other hand, the LOAD action will remove the TCS from the list of participating tasks so that the telescope will not receive any commands during the observation.

The JOS maintains a set of DRAMA parameters that are accessible to other tasks. The following parameters are available:

- Recipe text
- Recipe input parameters
- Recipe iterator values
- Configuration file name
- SETUP_SEQUENCE parameters
- SEQUENCE parameters

These parameters are monitored and displayed by the observation console task, described in the next section.

6. JCMT OBSERVATION CONSOLE

The JCMT Observation Console (JOC) is the graphical user interface through which the observer interacts with the OCS. It is implemented as a DRAMA task and is written in Perl/Tk. As shown in Figure 1, the JOC receives data from most of the tasks in the OCS. The JOC control path is through the JCMT_INST or JOS tasks. It is possible to have more than one instance of the JOC running, so that, for example, both the telescope operator and the astronomer can monitor the OCS from their own workstations.

Figure 4 shows a snapshot of the Observation Console in operation. The Observation Console window is divided into seven regions, each of which is detailed below.

A. Pull-down menu region. The menus shown in Figure 4 are set up for engineering tests. In this configuration, the menu provides the capability to load/unload instruments, configure the JOS, start observations, and load/unload tasks. During normal observing, only the File menu will appear in region A.

B. Control buttons/State Information. This region provides a set of buttons to allow the astronomer to control the progress of the observation. The buttons in this region send commands to the JOS to pause/continue/end/abort the observation. In addition, this region contains text labels that indicate the front-end currently in use, the running state of the JOS (e.g., RUNNING, ABORTING, etc.), and whether breakpoints have been enabled.

C. Recipe text. This region shows the recipe text in a scrollable window. In addition there is a highlight bar that shows the current instruction being executed.
D. Recipe Configuration. This region shows the name of the configuration file that defines the observing configuration currently being executed. In addition, the region shows the recipe parameters. When run in engineering mode the recipe parameters appear as editable text boxes so that the user can test the effect of changes to the recipe parameters. During normal observing, the recipe parameters will appear as regular text labels (not editable).

E. Recipe Execution. This region shows the progress of the observation. It is divided into three areas:

1) Recipe Iterators – shows the current iteration number for all loops in the recipe.
2) Setup Sequence Parameters – shows the parameters sent on the most recent SETUP_SEQUENCE command.
3) Sequence Parameters – shows the starting, current, and ending step numbers for the current sequence.

F. Task State. The OCS console monitors the state of all tasks participating in the recipe. This region shows, via green or red squares, whether the tasks have been Configured, Setup, or are “In Sequence”. This provides a visual indication if some task has gotten hung in a state, when all the other tasks have progressed to the next state.

G. Info messages. This is a scrollable text box that contains messages received from the various tasks involved in the observation. Informational messages are show in green and error messages are shown in red.

Figure 4. JCMT Observation Console
As currently implemented, the JOC does not have an interface to the observing queue task. This is mostly because we are currently in the development phase of the OCS and are not using the queue to perform engineering tests. However, a graphical user interface to the observing queue does exist, implemented as a Perl/Tk widget (Tk::OCSQMonitor) and making use of the DRAMA libraries to communicate with the other OCS tasks. At some future date, it is expected that the JOC will be modified to include an instance of Tk::OCSQMonitor. UKIRT currently uses Tk::OCSQMonitor as its user interface to the UKIRT observing queue task. Figure 5 shows the UKIRT queue monitor in operation. When implemented on JCMT, the queue monitor will look nearly identical to the UKIRT version.

![OCS Queue Monitor](image)

**Figure 5. OCS Queue Monitor**

In this example of the UKIRT Queue Monitor, we can see that two MSB’s have been submitted to the queue. Each MSB has two observations. The first observation of the first MSB has been completed and the second observation has been sent for execution, but not yet completed. The two observations for the second MSB are queued, but have not yet been sent for execution. When the first MSB is completed, the astronomer will be presented with a pop-up dialog box in which they will be asked whether the data taken for the just-completed MSB should be accepted or rejected. If they accept the data for the MSB, it will be removed from the queue. If they reject the data, the MSB will stay on the queue so that it can be re-observed, should the astronomer wish to do so.
7. CONCLUSION

The JCMT observing queue and recipe sequencer provide a solution to the complex problem of performing observations involving numerous telescope and instrument subsystems. The OCS has been developed with simple, standardized interfaces between tasks to minimize the communication complexity between the tasks that are involved in performing an observation. The OCS also makes use of software re-use concepts, particularly in the observing queue, because the same software will run at both UKIRT and JCMT\(^{10}\). The combination of the OCS, HARP-B, and ACSIS will greatly increase the scientific output of JCMT in the coming years.

REFERENCES