Chapter 3

Data Reduction Pipeline

In this chapter, we describe the data reduction pipeline that is used to convert the raw data logged by the telescope control system into light curves. This includes the software tools, the data filtering and calibration steps implemented by those tools, and the database system used to store the results of this pipeline for later scientific analysis. A major goal in the the design of the pipeline was to make it as automatic as possible, consistent with producing reliable, high-quality output data. This is important both due to the size of the monitoring program and to avoid opportunities for bias to enter the data set.

The reduction pipeline is organized into three levels of reduction, depicted in figure 3.1. This division is intended to avoid needless computational expense by storing intermediate data products at convenient points in the calibration process so that minor changes to the calibration procedures do not require a complete end-to-end recalibration. Additionally, the pipeline is designed to provide enough logging to ensure that a reduced data set can be reproduced in the future if needed, even if the standard parameters have been changed in the meantime. This also allows multiple reductions with different parameters to be stored in the database independently without requiring redundant copies of the intermediate data products.

The pipeline begins with a log file (VAX control system) or data archive (MCS control system) which contains the raw data stored by the control system. The *low-level* reduction script operates on these data to perform basic calibration steps that are unlikely to be changed often. Essentially, this reduction step consists of obtaining the results of the FLUX, CAL, and POINT procedures and filtering out those procedures that are completely unusable. In the current implementation, the CAL procedures are used to perform relative calibration as described in section 3.2.2.1 below, although this step properly belongs in the high-level calibration. The low-level scripts differ markedly in implementation between the VAX and MCS control systems, but both perform the same steps using essentially the same algorithms. The output of the low-level reduction is written into the reduction database in a common set of tables for both control systems. Further reduction steps are control-system agnostic—this is the only pipeline layer with different implementations for the two control systems.

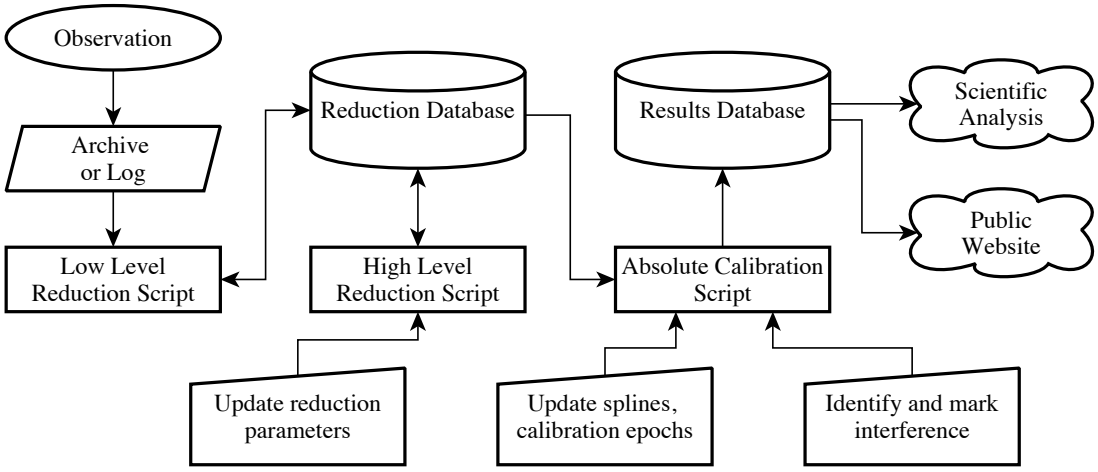

Figure 3.1. Overview of the data reduction pipeline.

The second stage in the pipeline is the *high-level* reduction script. In this stage, most of the flux density calibration steps are applied, including the gain curve and focus correction (section 2.1.2.3), and the error model (section 3.2.3). The output of the high-level reduction stage is written back to the reduction database.

Finally, the *flux density calibration* script convert the results of the high-level reduction into the fully calibrated end product of the pipeline. This script applies the data flags to remove unreliable measurements (section 3.2.1), uses the astronomical calibrator (3C 286) to provide calibration of the flux density scale, and applies the calibrator variation spline (section 3.2.2.2) and nonthermal error correction factor (section 3.2.3.2) to the data. The output of this script is a set of comma-separated-value (CSV) files containing the fully calibrated flux density measurements for each source. These files are then used to update the results database and for scientific analysis.

The storage requirements for a monitoring program of this size present some challenges. The volume of data, while not enormous by modern standards, is sufficiently large that careful organization is needed. This is especially true for a long-term monitoring program like this. Because the data will be reduced on approximately a monthly basis, an initial investment in proper organization of data and code yields a repeated benefit. To achieve this goal, two database systems—one for storing the intermediate data reduction products and metadata describing the reduction process and one to contain the processed data—have been developed.

This chapter is organized as follows. In section 3.1, we introduce the software tools used to implement this pipeline. In section 3.2, we describe in detail the steps that make up the reduction pipeline and discuss their implementation. Finally, in section 3.3, we describe the organization of the database that stores the results of the pipeline and makes them available for analysis and to the outside world. Additional details about the software and database structure may be found in appendixes A and B.

3.1 Software Tools

There is certainly no shortage of numerical software and programming languages suitable for data analysis and reduction. For this project, we have chosen to use the Python[1] programming language. Python is an object-oriented, bytecode-interpreted language that is suitable for either scripted or interactive use. It is supported on a wide variety of operating systems including Linux, Apple OSX, and Microsoft Windows, making it convenient for use in a mixed computing environment. The SciPy[2] and NumPy[3] libraries provide a wide variety of scientific and numerical methods that make Python especially appropriate for scientific computing. High-quality plotting suitable for either interactive use or publication plots is available using the Matplotlib[4] library (Hunter 2007). A wide variety of Python library modules are available, making it convenient to interface with other software. Additionally, Python is increasingly popular for astronomical data analysis, with projects such the National Radio Astronomy Observatory's (NRAO) Common Astronomy Software Applications (CASA) using it to provide a convenient user interface.[5]

Regardless of the choice of programming language and environment, a library of functions to implement the particular data handling and calibration routines is needed. We have implemented most of these in a Python module named *Arcreduce*, described in section 3.1.2.2 below. A set of utility routines needed by the reduction scripts and by Arcreduce, called *py40m*, has also been developed. The py40m module contains routines for interacting with the database, performing the actual calibration computations, and includes a number of convenience routines for, e.g., converting between date formats.

As described in section 2.1.1.1, data in this program were recorded using two different telescope control systems. The VAX and MCS control systems differ greatly in their capabilities, reflecting the enormous increase in computing power and data storage capability available to the newer system. The data interface hardware used for both systems is the same, with radiometer sampling and telescope monitoring data recorded through the Universal Back-End (UBE). In particular, both systems receive radiometer samples at 1 kHz, normally Dicke switched to provide an effective 500 Hz switched sampling rate. The VAX control system, due to its limited bus speed, normally did not record the full stream of samples, instead integrating these in software and storing a summarized result of the procedure to disk for analysis. Between scheduled procedures, no radiometer data were recorded. The MCS control system, on the other hand, permanently records every 1 ms sample, regardless of whether an active observation has been commanded. A human-readable log including on-line integrations for quick debugging is also provided, but the off-line analysis software is responsible for converting the raw samples into procedure results for science data. The MCS control system also stores continuous records of telescope monitoring data, including its pointing, thermometer readings, weather station outputs, etc., whereas the VAX control system only stored relatively low-rate snapshots of these status data.

[1] `http://www.python.org`
[2] `http://scipy.org`
[3] `http://numpy.scipy.org`
[4] `http://matplotlib.sourceforge.net`
[5] `http://casa.nrao.edu/`

The VAX control system stored approximately 1 MB of data per day. The MCS control system stores a bit more than 1 GB per day—more than a three order of magnitude increase. The massive increase in available data has many effects. It enables much finer study of the telescope's behavior for debugging, and is particularly better at identifying rare and unpredictable errors because all the data are available at all times, without the need to enable a debugging mode at the right moment. The increased rate also increases the amount of storage by a factor of 1000. Although advances in storage technology have made this a negligible cost increase, the data management requirements (e.g., arranging to back up the data regularly) are non-trivial. Last, and most relevant to this chapter, the two control systems require very different reduction software. In particular, the VAX control system performed the step of translating radiometer samples into procedure results, whereas the MCS control system leaves this to the reduction software layer.

3.1.1 CMBPROG and the VAX Control System

For the first two years of the program, the telescope was controlled using the VAX control system. In this section, we briefly describe the VAX control system and CMBPROG, the analysis tool used to process the data.

3.1.1.1 VAX Control System

The VAX control system was used to control the 40 m telescope from the early 1990s until August, 2010. A full description of the control system is available in Pearson (1999), here we briefly summarize that description. The telescope hardware was controlled by a single-board DEC MicroVAX *satellite computer* located in the pedestal of the 40 m, called OV40M. OV40M was equipped with analog-to-digital and digital-to-analog interfaces as well as a parallel data bus that enabled it to communicate with and control the telescope hardware. One component of the control software stack, called the *satellite program* or SAT, executed on OV40M. This program included processes for handling servo controls, radiometer data collection, logging, and command and response processing for interacting with the other layers in the software stack. For reliability, OV40M was not equipped with a hard disk. Instead, it booted via Ethernet from a boot image on another MicroVAX, OVCM, which was located in the control building.

In addition to hosting the boot image for the satellite computer, OVCM also executed the two other programs in the control program software stack: the *command and control program* (CCP) and the *terminal interface program* (TIP). The TIP was a terminal-based user interface that allowed commands to be sent to the telescope, schedules to be queued, and the status of the telescope to be monitored. Multiple copies of the TIP could be executed simultaneously, enabling several users to monitor the telescope. The CCP acted as an intermediate layer between the TIP and the SAT programs, processing and serializing commands from multiple TIPs, dispatching responses to the appropriate TIPs, processing schedules, and writing data to the output logs.

A simple, reasonably general scripting language was implemented in the CCP. This included control commands, looping constructs, and subroutine handling. An example observing script is shown in Listings 3.1 and 3.2.

3.1.1.2 Overview of CMBPROG

To analyze the data stored in the log files from the VAX control system, the CMBPROG program was developed by Erik Leitch using a framework written by Martin Shepherd for the DIFMAP package (Leitch 1998; Shepherd 1997). CMBPROG provides a parser to load the data from the log files generated by the CCP, a graphical user interface for plotting and interactively editing the data, and facilities for automated analysis and filtering of the data.

Data within CMBPROG are organized in a series of data structures called *Procedures*, each representing the time series of results of the executions of a particular radiometry or observation-related procedure. Each Procedure contains a set of *Members* containing the actual data recorded by the CCP. For example, the output of the execution of a FLUX measurement is stored in the FLUX Procedure. The actual measured flux densities of a series of flux density measurements is stored in the flux Member, which is referred to as FLUX.flux.

Procedure results can be filtered based on various criteria, e.g., to restrict the results to a particular source. Search criteria are implemented via a data structure called a *ferret*.[6] Several ferrets can exist in parallel, allowing the results for different selection criteria to be handled separately. The ferret also provides a mechanism for reconciling the time bases of different procedures. The results of one procedure may be *referenced* onto the time base of another within the context of a ferret. Several referencing algorithms are implemented. Referencing can be used to associate, e.g., the most recent previous wind measurement with a POINT procedure with that POINT, or the average of all CAL procedure flux measurements within a two-hour window around a FLUX procedure with that FLUX.

A wide variety of data filtering and calibration procedures are provided by CMBPROG. CMBPROG is best suited for interactive use, but does provide a scripting language for batch processing. An excerpt from the data reduction script is given in Listing 3.3. CMBPROG can export processed data in a log file format similar to the log files written by the CCP, or by writing the contents of selected Members to a text file.

3.1.1.3 Retiring CMBPROG

Although CMBPROG is a powerful and useful tool for data reduction, because it consists of a monolithic compiled C program, it is somewhat cumbersome to add features. Also, many of its plotting and numerical computing features are now available as libraries to the general-purpose Python programming language, reducing the need for a custom package. In addition, only a relatively small subset of the functionality provided by CMBPROG is actually used for the data reduction pipeline. For these reasons, we chose not to

[6]The name “ferret” is derived from the phrase “to ferret out,” meaning to search tenaciously for and find something.

```
subfile doflux
!====================================
!      CGRABS Flux measurement Schedule
!      04jan10_0_0.sch
!====================================
  anwait
  weather
  setfocus
  setfocus
  flux samples=8 reps=1
  tag
endsubfile

subfile dopoint
  anwait
  setfocus
  setfocus
  flux samples=8 reps=1
  setfocus
  setfocus
  record tilt
  poff 0.0 0.0
  setfocus
  setfocus
  point samples=10 reps=1
  record tilt
  setfocus
  setfocus
  point samples=10 reps=1
  record tilt
  setfocus
  setfocus
  point samples=10 reps=1
  record offsets
  zero
  weather
  setfocus
  setfocus
  flux samples=8 reps=1
  cal reps=1 diode=cal
  cal reps=1 diode=noise
endsubfile

subfile calib
  anwait
  setfocus
  setfocus
  flux samples=10 reps=1
  setfocus
  setfocus
  record tilt
  poff 0.0 0.0
  record tilt
  setfocus
  setfocus
  point samples=8
  record tilt
  setfocus
  setfocus
  point samples=8
  record tilt
  setfocus
  setfocus
  point samples=8
  record offsets
  setfocus
  setfocus
  ave
  weather
  setfocus
  setfocus
  flux samples=10 reps=1
  cal reps=1 diode=cal
  cal reps=1 diode=noise
  zero
endsubfile
```

Listing 3.1. Example VAX control system schedule. This excerpt defines the compound procedures for performing a FLUX, CAL, or POINT procedure.

```
subfile region_120
obs/nowait J2331-1556
sch/sub/nowait/lst dopoint start=21:34:29 stop=01:25:17
obs/nowait J2336-1451
sch/sub/nowait/lst doflux start=21:34:29 stop=01:40:20
endsubfile

subfile region_119
obs/nowait J0116-1136
sch/sub/nowait/lst dopoint start=22:49:45 stop=03:40:45
obs/nowait J0111-1317
sch/sub/nowait/lst doflux start=22:54:46 stop=03:25:42
obs/nowait J0110-0741
sch/sub/nowait/lst doflux start=22:24:40 stop=04:00:50
obs/nowait J0110-0415
sch/sub/nowait/lst doflux start=22:04:35 stop=04:15:53
obs/nowait J0127-0821
sch/sub/nowait/lst doflux start=22:44:44 stop=04:10:52
obs/nowait J0125-0005
sch/sub/nowait/lst doflux start=22:04:35 stop=04:45:59
obs/nowait J0141-0202
sch/sub/nowait/lst doflux start=22:29:41 stop=04:56:01
obs/nowait J0141-0928
sch/sub/nowait/lst doflux start=23:04:48 stop=04:20:54
obs/nowait J0140-1532
sch/sub/nowait/lst doflux start=23:39:55 stop=03:40:45
obs/nowait J0132-1654
sch/sub/nowait/lst doflux start=23:44:56 stop=03:20:41
endsubfile
```

Listing 3.2. Example VAX control system schedule. Here, the subfiles shown in Listing 3.1 are called to perform the observations for a series of regions. Two such regions are shown here.

```
!! chop low-snr points (using a random flag)
clip point.snr, 0, 2, inc, swp

!! flag failed points
setintd 4800
pointflag flux
useflags -8


!! cut out saturated / abnormal fluxes
!! This is a fairly low threshold -- not truly "saturation", but
!! grabs some periods of anomalously high TP also.
clip flux.atp, 10000, 50000, exc, flux
clip flux.btp, 10000, 50000, exc, flux
clip flux.ctp, 10000, 50000, exc, flux
clip flux.dtp, 10000, 50000, exc, flux

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! Now clean up the calibration diode before we use it
!!
@/home/bigjoe/caltech/glast/pipeline/cmbprog/calcuts.cmb

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! Now let's apply the calibration diode to put various
!! epochs on equal scales before doing any outlier filters.
!!

!! Create a ferret with cals for all sources
diode CAL
caltag ANT
make cal, allcal

!! do the calibration with the CAL diode
diode CAL
setcalt box, false
setcalb 7200, 2
setcald 0, 0.1
setcaln allcal
! tcal_ant in Jy (divide by 2 since it's only on half the time in a CAL)
telpar "tcal_ant", 8.33 / 2
calib flux, flux

setintt prev, false, true
setintd 3600, 0.1
reference flux.time, setfocus.cz, focus

!! Add some aliases
source ""
make pcor, allpcor
make poffsets, allpoff
make point, allpoint
```

Listing 3.3. Main CMBPROG calibration code, executed after loading the data from the log files and before dumping the data for each source into text files for import into Python.

extend CMBPROG to support data from the MCS control system. Instead, we have reproduced the essential elements of CMBPROG as a Python module and used this as the basis for reducing data after the control system upgrade.

For data recorded under the old control system, however, we continue to use CMBPROG for reduction. Because this is done in the low-level reduction stage, these early data are rarely re-reduced, so the CMBPROG stage is executed only occasionally. When changes that affect the low-level reduction are made, care must be taken to ensure that the CMBPROG scripts are consistent with any changes to the Python scripts that implement the calibration procedures for the newer data (see section 3.1.2).

3.1.1.4 Interface between CMBPROG and Python

Although CMBPROG was not designed to be executed as part of another program, it is possible to execute it from within a Python script by issuing commands over a pipe to its standard input. For this purpose, we implemented a set of routines in `py40m.cmbprog` to execute and control an instance of CMBPROG. Within the low-level reduction script, this interface is used to load the raw data from the telescope log files and execute the CMBPROG calibration scripts. The output for each source is then written to a series of text files and loaded into Python. The data from the text files are then stored in the reduction database for further processing.

3.1.2 Python, Arcreduce, and the MCS Control System

We now turn our attention to the newer MCS control system and the software tools for reducing the data collected with it. We first give an overview of the new control system hardware and software, then describe the Arcreduce Python module that is used to manage those data.

3.1.2.1 MCS Control System

A full description of the MCS control system architecture, focused on the implementation of the drive system, is given in Shepherd (2011a). An online manual for the control system, including a description of the scripting language, user interface, and other details, may be found at Shepherd (2011b). In this section, we give a brief overview of the control system hardware and software.

Like the VAX control system, the MCS control system divides the critical telescope control tasks and user interface tasks between separate networked computers. This architecture is shown in figure 3.2. The *control computer* is responsible for managing the telescope operation, including all time-critical operations. It runs a real-time variant of the Linux operating system and is equipped with analog and digital input/output cards connected to the telescope hardware. The *observing computer* runs an ordinary (not real-time) Linux distribution. Communication with the control computer is managed by the control program, *telcontrol*. The user interface for queuing schedules, executing commands, and monitoring the telescope and radiometer

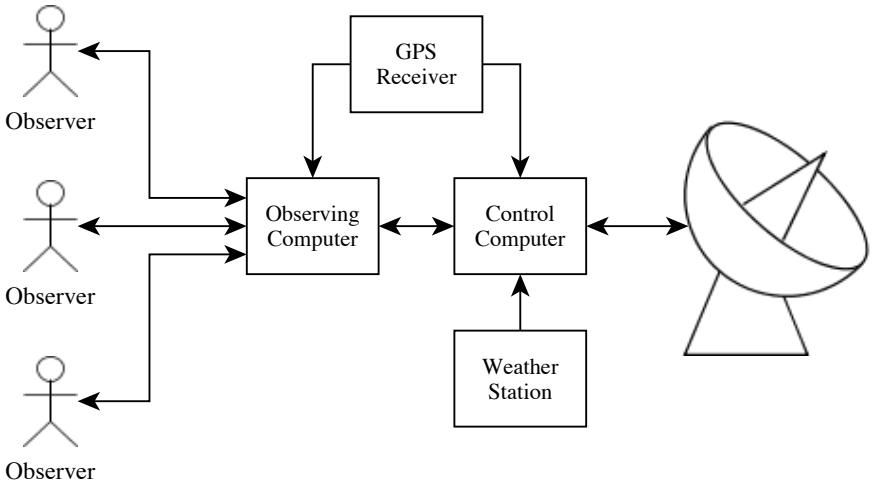

Figure 3.2. Schematic overview of the MCS control system architecture. The global positioning system (GPS) receiver provides a pulse-per-second signal to keep the clocks on the observing and control computers precisely synchronized with UTC.

status is provided by the *telviewer* program. To access the interface, observers connect to the observing computer via a remote desktop connection and execute instances of the telviewer program.

The MCS control system supports a schedule scripting language that is more general than that provided by the VAX control system. The MCS control system provides script-level access to low-level telescope control commands. Functions to implement radiometry procedures are implemented as script-level functions, rather than having hard-coded implementations defined in the control program. Listings 3.4 through 3.6 show a few examples of excerpts from MCS control system scripts.

3.1.2.2 The Arcreduce Python Module

We have implemented a Python module, Arcreduce, which contains the classes and methods needed for reducing MCS control system data. The `ArchiveReader` class is the engine for converting the low-level radiometer samples into procedure results. The replacement pieces of CMBPROG were implemented as the `CalManager` class and its related methods and classes. The Arcreduce module is described in detail in appendix A, here we give a brief overview of the software.

The `ArchiveReader` class provides a flexible engine for processing data from the MCS control system data archives. Figure 3.3 shows a schematic view of the architecture of the `ArchiveReader` engine. The user first instantiates an `ArchiveReader` object to manage the data decoding process, specifying the start and end dates for the reduction. The `ArchiveReader` opens a connection to the data archive using the readarc module.[7] The user then instantiates decoder objects and connects them to the `ArchiveReader`. Each decoder class is a subclass of the `GenericDecoder` class, which defines the interface between the

[7] `http://www.astro.caltech.edu/~mcs/ovro/40m/help/readarc.html`

```
# Arrange for the telescope to be stowed when this schedule ends.
at_end {
  stow
}
# Get the definitions of the standard procedures.
import "$TCS_SCHED_DIR/schedlib.sch"
# Load fermi-specific procedure definitions.
import "$TCS_SCHED_DIR/fermilib.sch"
#-----------------------------------------------------------------------
# Define a set of regions on the sky. Each parameters of each region
# are as follows:
#  String name,          # The name of the region.
#  Double start,         # The earliest LST (hours) to observe this region.
#  Double stop,          # The latest LST (hours) to observe this region.
#  Source psrc,          # The source to peak up the pointing on.
#  Double point_dt,      # The pointing integration time.
#  listof Source srcs    # The list of sources whose fluxes are to be measured.
#-----------------------------------------------------------------------
listof SourceRegion regions = {
  {"region_103", 23:30:29, 00:29:30, J2323-0317, 0:0:10,
   {PKS2320-021, J2335-0131, J2337-0230, J2301-0158, BBJ2247+0000, J2247+0310,
       J2257+0243}
  },
  {"region_120", 23:52:29, 00:52:01, J2331-1556, 0:0:10,
   {J2336-1451}
  }
}

# Start the observations, passing the following function the list of regions
# to be observed, followed by the allowed elevation range.

observe_fermi_sources $regions, 30, 70
```

Listing 3.4. Example MCS control system schedule. This is an example of a program observing schedule. It defines a number of regions and their pointing calibrator sources. In a full schedule, many more regions would be defined.

```
command flux(Count reps, Double integ_dt, Double idle_dt, Double xoff,
             Boolean both) {
  Double flux_a = 0.0
  Double flux_b = 0.0
  Double flux_c = 0.0
  Double flux_d = 0.0
  Double sdev_a = 0.0
  Double sdev_b = 0.0
  Double sdev_c = 0.0
  Double sdev_d = 0.0

  Double dx = $xoff
  TrackingOffsets old = $tracking_offsets()
  Boolean ok = false
  at_end {
    if(!$ok) {
      mark one, f0+f1    # Use feature markers 0 and 1 to indicate failure.
    }
    label "none"
    offset x=$old.x    # Restore the original X offset.
    print "Procedure FLUX ended"
  }
  print "Procedure FLUX starting"
  catch $signaled(source_set) | !$drives_enabled() | $signaled(suspended) |
      $aborted() {
    while($iteration < $reps) {
      label "flux:A"       # Perform the measurement at the +ve off-source
          position.
      flux_integ $integ_dt, $idle_dt, $old.x + $dx, true, true, $flux_a, $sdev_a
      label "flux:B"      # Perform the first on-source measurement.
      flux_integ $integ_dt, $idle_dt, $old.x, true, false, $flux_b, $sdev_b
      label "flux:C"      # Perform a second on-source measurement.
      flux_integ $integ_dt, 0.0, $old.x, false, false, $flux_c, $sdev_c
      # In two-sided mode, toggle the sign of the offset.
      if($both) {
        dx = -$dx
      }
      label "flux:D"      # Perform the second off-source measurement.
      flux_integ $integ_dt, $idle_dt, $old.x + $dx, true, false, $flux_d,
          $sdev_d
      print "Fluxes A=", $format_double(".4f",$flux_a),
                    " B=", $format_double(".4f",$flux_b),
                    " C=", $format_double(".4f",$flux_c),
                    " D=", $format_double(".4f",$flux_d)
      print "Flux (B+C-A-D)/2 = ", $format_double(".4f",($flux_b + $flux_c -
          $flux_a - $flux_d)/2.0), " +/- ", $format_double(".4g",$sqrt($sdev_a*
          $sdev_a + $sdev_b*$sdev_b + $sdev_c*$sdev_c + $sdev_d*$sdev_d))
      mark one, f0      # Indicate that this iteration of the procedure was
          successful.
    }
    ok = true    # Indicate that the procedure ran to completion.
  }
}
```

Listing 3.5. MCS control system schedule for performing a FLUX procedure.

```
#-----------------------------------------------------------------------
# This command is part of the flux() command. It moves the telescope
# to a given absolute X offset, and if this takes less than
# $idle_dt, waits until $idle_dt hours have passed. It then performs
# an integration of $integ_dt hours.
#-----------------------------------------------------------------------
command flux_integ(Double integ_dt, Double idle_dt, Double xoff, Boolean move,
                   Boolean refocus, Double flux, Double sdev) {
  # Change the tracking offset?
  if($move) {
    offset x=$xoff
    # Wait for longest of the acquisition time and the idle time.
    until $acquired(source) & $elapsed >= $idle_dt
  }
  if($refocus) {
    focus
    until $acquired(focus)
  }
  # Integrate for the specified length of time.
  request_flux $receiver, switched, $integ_dt
  until $acquired(flux) | $elapsed >= $integ_dt+0:0:5

  # Get the returned flux.
  flux = $requested_flux.flux
  sdev = $requested_flux.sdev
}
```

Listing 3.6. MCS control system definition of the `flux_integ()` routine used by the FLUX procedure. In the VAX control system, functions at this low level were defined within the control program, rather than implemented through schedule scripts.

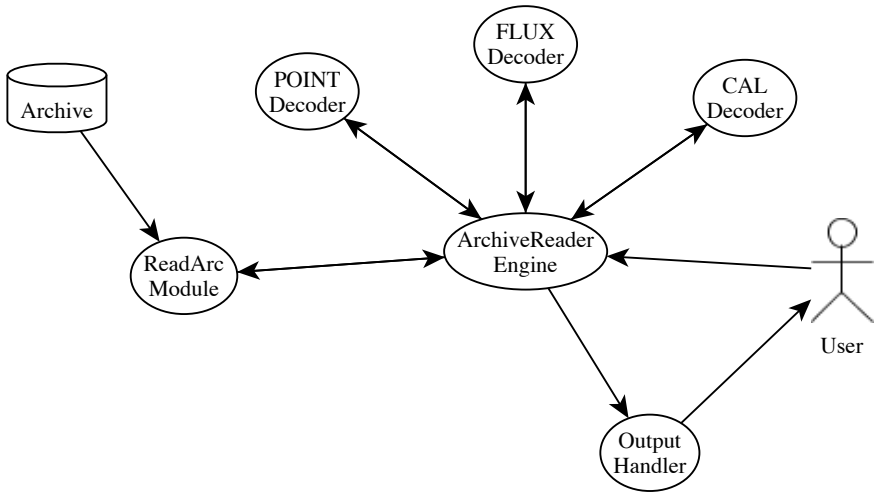

Figure 3.3. Architecture of the `ArchiveReader` engine. The user or reduction script instantiates and interacts with an `ArchiveReader` object. This object uses the readarc module to obtain raw data from the archive. These data are passed frame-by-frame to a set of decoder objects, each of which is responsible for identifying and processing a single type of radiometry procedure. When the decoder finishes decoding a procedure, it passes the result to the `ArchiveReader`, which notifies one or more output handler objects of the result.

decoder and the `ArchiveReader`. The class hierarchy for the decoder objects is shown in figure 3.4. Finally, the user instantiates and connects one or more output handler objects to the `ArchiveReader`.

Data in the archive are organized into named registers, which are stored in data frames, one frame per second. For data collected at a higher rate, such as the radiometer samples (1 kHz) or the mount encoder readings (10 Hz), array registers allow multiple data points to be stored in a single frame. The `ArchiveReader` must specify to the readarc module which registers are to be read from the archive. By default, it provides the real-time clock time stamp for each frame, the current source name register, and the frame label (a free-form string register that is used to identify the currently active radiometry procedure). When a decoder is first connected, it passes a list of the registers for which it requires data to the `ArchiveReader`, which then configures the readarc module to read those registers if they are not already active.

Once the `ArchiveReader`, decoders, and output handlers have been set up, the user or reduction script repeatedly calls the `handle_frame()` method until it indicates that the end of the requested date period (or the end of the data archive) has been reached. For each frame, which corresponds to 1 s of data, the `ArchiveReader` reads the data from the archive using readarc, then notifies each decoder object of the new data. The decoder objects each implement a state machine to identify and process the data for a radiometry procedure. When the end of a procedure is encountered, the decoder returns the results of the procedure to the `ArchiveReader` object, which in turn passes this result to any connected output handlers.

Although several output handlers are implemented, the `CallbackOutputHandler` is normally used during data reduction. When a procedure is completed, this output handler calls a user-specified function

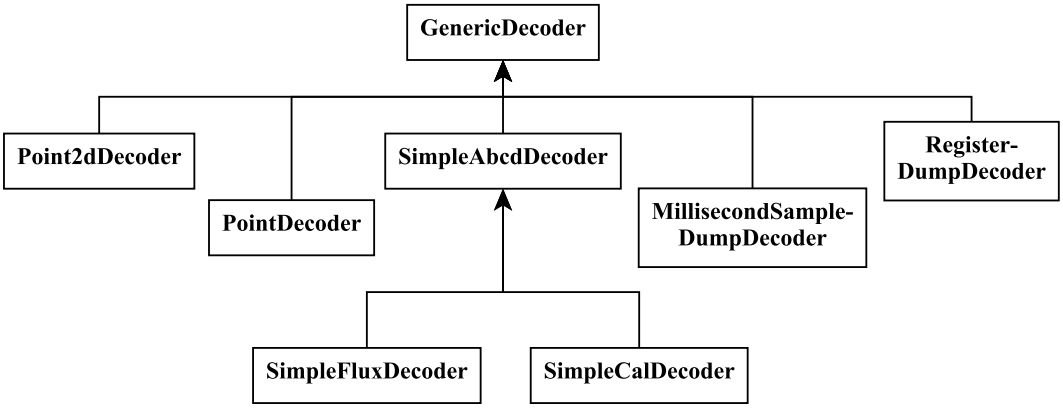

Figure 3.4. Class hierarchy for procedure decoders in the ArchiveReader module. The `SimpleAbcdDecoder` provides generic decoding of radiometry procedures that execute an A-B-C-D double-switching pattern. The `RegisterDumpDecoder` and `MillisecondSampleDumpDecoder` are "phony" decoders—these are used for dumping raw 1 s or 1 ms samples to a file, rather than for decoding procedures.

with the the results of the procedure. This is normally used with the `CalManager` class, which provides a callback function that stores the results of each procedure in the appropriate `Procedure` object, described in section A.1.

3.1.2.3 Reducing Radiometry Procedure Data

The MCS control system provides access to a much greater amount of observation data than did the VAX control system, but this additional detail comes at a cost: decoding the results of an observation requires a much more sophisticated analysis. Whereas the VAX control system software processed and reported the results of each radiometry procedure, requiring only parsing the output logs to determine the results, the MCS control system stores the raw radiometer samples, which must be processed. This greater complexity is undoubtedly worthwhile, however, as it permits more careful scrutiny of the telescope performance, and enables improvements in the procedure reduction algorithm to be applied retroactively because all the data, not just the summarized final result, are available.

The `ArchiveReader` class supports decoding three radiometry procedures: FLUX, CAL, and POINT. The first two procedures are very similar, so these are implemented using a single set of logic with very minor adjustments. The POINT procedure is quite different, and must be handled in both the original and "2D" implementations, as described in section 2.2.2. The decoders for these procedures currently use the 1 s averaged radiometer data available in the data archive rather than working with the full-rate millisecond samples.

FLUX procedure. As described in section 2.2.2, the FLUX procedure consists of four integrations, labeled ξ_{A}, ξ_{B}, ξ_{C}, and ξ_{D}. The control system labels the 1 s data frames with the strings "FLUX:A" through

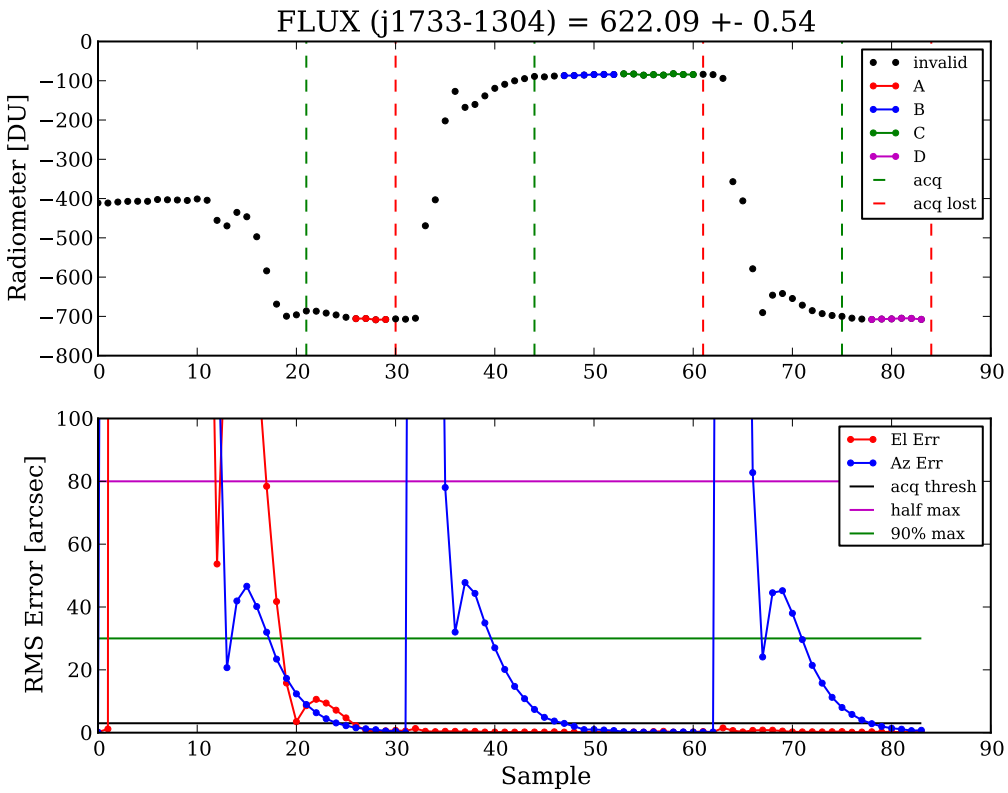

Figure 3.5. Illustration of the FLUX procedure decoder. *Top:* Dots show the uncalibrated 1 s average difference between the *ant* and *ref* inputs. The valid samples that are integrated for the A, B, C, and D segments of the procedure are indicated. The vertical dashed lines indicate where the control system signaled that the source being observed had been acquired or when acquisition was lost. *Bottom:* Red and blue data show the elevation and azimuth pointing error. The horizontal lines indicate the offset from the nominal Gaussian beam profile corresponding to the half power and 90% power points, and the acquisition threshold used by the software decoder.

"FLUX:D" to enable the decoder to identify the starts and ends of FLUX procedures as the archive is processed. The *tracker* thread within the control system produces an *acquired* flag that indicates when the tracking servo has first acquired the target source. Relying solely on this flag is not an adequate measure to reject samples where tracking errors are significant. In the example shown in figure 3.5, just after the acquisition signal is raised (indicated by the first vertical line in the upper plot), an elevation tracking glitch occurs, resulting in a clear change in the radiometer output. To ensure that only data collected with accurate pointing are included in the integrations, the decoder identifies the last continuous run of samples with tracking errors less than $3''$ in each segment and integrates only these samples. Using the selected samples in each segment, the mean and standard deviation are computed, and these are combined to produce the FLUX result.

The control system records an estimate of the result of the FLUX procedure in a human-readable log file as the data are collected. This is useful for debugging, but because it uses a less cautious decoder, its results

are somewhat less accurate than those from the `ArchiveReader` decoder. A comparison between the results in the log file and those obtained from `ArchiveReader` finds that the flux density measurements reported in the log file are systematically lower than those computed by `ArchiveReader`. The median difference is about $0.9 \times \sigma_{15}$, a rather large disagreement. This is due to the inclusion of less accurately pointed data in the online integrations.

CAL procedure. The CAL procedure is nearly identical to the FLUX procedure so the same decoder described above is used for the CAL procedure as well. Because the CAL procedure requires a slew only before the A segment (when it slews off-source), the tests for accurate tracking do not typically discard data except during the A segment.

POINT procedure. The decoding requirements for the POINT procedure are different than for the FLUX and CAL procedures. In addition to the obvious difference in the data collected, the result of a POINT procedure is required in real time—the pointing offset must be computed as the data are taken so that it can be used to point the telescope accurately. The control system performs this calculation in real time and, following a successful POINT, indicates the result by moving the telescope to the measured offset and raising a flag. If a POINT fails, this flag is not raised, thus indicating the failure to the decoder. During later analysis, the `ArchiveReader` decoder has no need to repeat the fitting procedure. Instead, the result indicated by the control system is simply adopted.

The `ArchiveReader` decoder does, nonetheless, integrate the samples within the segments of the POINT (or POINT2D) procedure. An example of the samples collected during a POINT2D procedure is shown in figure 3.6. Using the integrated averages and estimated uncertainties together with the fitted position indicated by the control system, the decoder fits a fixed-beamwidth Gaussian beam profile to the results to determine the flux density of the source and to compute the signal-to-noise ratio. These results are used to filter bad pointings out of the data stream.

3.2 Reducing the Data

We now turn to the specific procedures used for reducing the data. These steps are implemented using the software tools—CMBPROG and ArcReduce—described in the previous section. In this section, we begin with the reduced procedure results, assuming that ArcReduce has already processed the raw samples for the MCS control system data. In this section, we organize the calibration steps into groups by function, first discussing data editing and filtering, then flux density calibration, and finally error estimation. In the pipeline, for convenience these functions are not necessarily performed in that order, but the end result is the same as if they were.

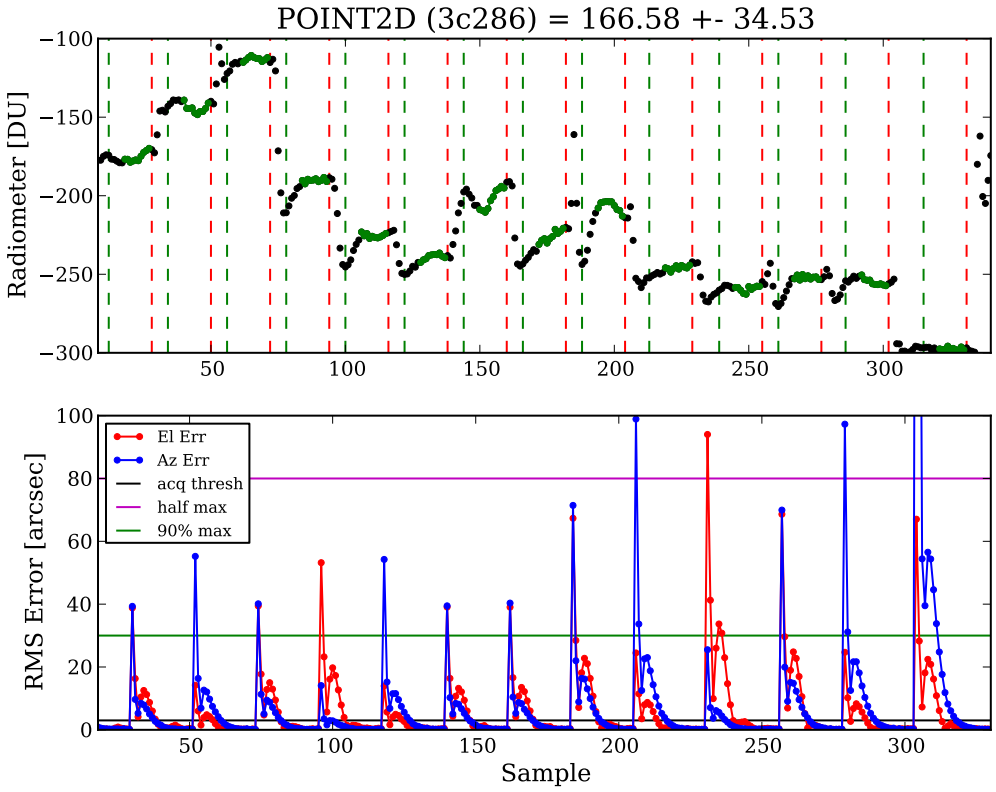

Figure 3.6. Illustration of the POINT2D procedure. *Top:* Dots show the uncalibrated 1 s average difference between the *ant* and *ref* inputs with the valid samples used for each integration indicated. Vertical lines indicate acquisition and loss of acquisition at the start and end of each segment. The segments are arranged in order as numbered in figure 2.30. *Bottom:* Azimuth and elevation tracking errors during the procedure.

3.2.1 Data Editing and Filtering

We first describe the data editing and filtering used to remove unreliable data. Parts of this procedure are performed in the low-level reduction scripts, but most filters are applied as part of the final flux density calibration procedure. This is done so that the suspect data remain in the database for later examination to evaluate whether the filters being applied are working as expected. Data filtered out early in the calibration process are inconvenient to evaluate later, so this is only done for data so obviously corrupt that access to them is unlikely to be needed often. This includes data lost due to pointing offset measurement and calibration diode reading failures.

Editing and removal of corrupted data is performed using both automated and manual filters. Where possible, we have used automated filters both to avoid laborious examination of data and to eliminate opportunities for bias to enter the pipeline. Care is taken to avoid flagging data based on criteria that could induce bias. For example, no flagging based on the flux density of a measurement is implemented.

Data to be edited or filtered are marked with *flags*. The flux density calibration script accepts a list of flags to either require or reject for each data point. Some of the flags require parameters—these are specified as reduction parameters which are stored as described in section 3.3.4 . In each section, we specify the flag name and the names of any parameters that affect it. The database tables that define the flags are described in section 3.3.4 and a full list of the flags is given in section B.2.

3.2.1.1 Date Interval Cuts

A list of date intervals for which all data should be ignored is maintained. This is the mechanism by which manual data editing is implemented. In addition, telescope calibration and maintenance periods, observations other than the monitoring program, and any other periods during which the data should not be included in our results are indicated with an entry in the list. The reduction database permits multiple sets of flagged date intervals to be stored simultaneously. The date interval cuts are enabled by excluding the `dateflag` flag and the set of flagged intervals is specified using the `DATEFLAG_SET_NAME` parameter.

In a few cases, anomalous calibrator source readings are dropped using a date interval cut. This was done frequently with 3C 161, which sometimes experiences anomalous pointing failures (see section 4.1.3 for further discussion). This sort of data editing is only used when there is very strong evidence for a pointing failure or other data contamination. Most frequently these edits are used for calibration sources that are expected to exhibit stable flux densities.

3.2.1.2 Wind, Sun, Moon, and Zenith Angle Cuts

Under high winds there is a systematic reduction in observed flux densities due to mispointing and poor tracking. Observations made when the instantaneous wind speed exceeds 15 mph ($6.7\ \mathrm{m\,s^{-1}}$) are discarded. A FLUX procedure is rejected if the wind speed exceeded the threshold either during that FLUX procedure

or during the POINT procedure that was used to obtain the pointing offset for that FLUX. To protect the telescope a "wind watchdog" program stows the telescope to a safe position pointing near zenith. For more details, see section 2.1.1.4. If wind data are not available, due to a weather station or logging failure, data are discarded.[8] Under the VAX control system, wind measurements were recorded with each POINT procedure, or at least about once per hour. Under the MCS control system, the weather station outputs are logged every second, but only one reading per minute is used by the reduction pipeline for wind speed flagging. To reject data flagged due to high wind, the `wind` flag must be enabled and the wind threshold in mph must be specified through the `WIND_THRESH` reduction parameter.

Observations at zenith angles less than 20° are discarded because the telescope is unable to track fast enough in azimuth to match the sidereal rate near zenith. The scheduling algorithm avoids scheduling sources for observation at these zenith angles, so few observations are lost. Observations at solar or lunar elongations less than 10° are also discarded. The scheduler does not avoid these areas of the sky so a small number of observations are lost. The zenith angle cut is enabled using the `za_limit` flag with a upper and lower zenith angle thresholds set with the `ZA_LIMIT_MIN` and `ZA_LIMIT_MAX` reduction parameters, which are normally set to 20° and 90°, respectively. The solar and lunar elongation limits are enabled via the `sun_angle` and `moon_angle` flags, with the thresholds set by the `SUN_ANGLE_THRESH` and `MOON_ANGLE_THRESH` parameters.

3.2.1.3 Pointing and Calibration Failures

An observation is rejected if a pointing offset was not obtained within the prior 4800 s, or if the pointing offset measurement immediately preceding the observation failed. FLUX procedures affected by these pointing failures are dropped during the low-level calibration process. Occasional scheduling errors resulted in observations without adequately measured pointing offsets. These observations are discarded. These flags are enabled by requiring the flag and by excluding the `pointing_model_only` and `pointtest` flags. During 2009 and 2010, FLUX observations of the pointing calibrators were performed just prior to measuring the pointing offset. These were used to evaluate the pointing performance and are excluded from our normal data set. These "point test" FLUX observations are identified by the presence of a POINT on the same source as a FLUX, within a short time period specified by the `POINT_TEST_MAX_DELAY` parameter. Finally, the `pdist` flag will identify FLUX procedures for removal if they were observed more than `PDIST_THRESH` degrees on the sky away from the preceding POINT measurement.

An observation is rejected if fewer than two reliable calibration procedures using the CAL diode were successfully executed within a two-hour interval centered on the time of the observation, or if the difference between the largest and smallest CAL diode measurement within that interval differ by more than 10%. FLUX procedures affected by this are dropped during the low-level calibration process.

[8] In the data published with Richards et al. (2011), data were accepted when wind data were not available. This change has caused a few data points to be rejected that were accepted in that published data set.

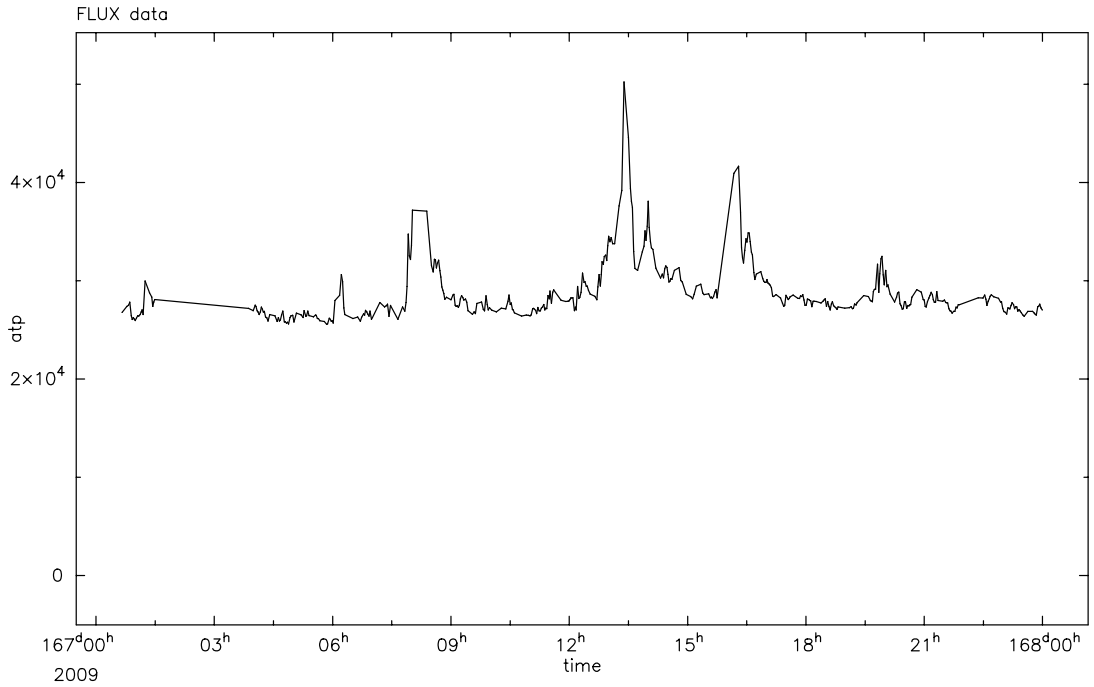

Figure 3.7. Average total power during the A segments of the FLUX procedures recorded on 16 June 2009. The strong spikes resulted from interference from inclement weather.

3.2.1.4 Saturation and Total Power Anomalies

The total power varies depending on the attenuator setting, receiver gain fluctuations, atmospheric conditions, and the observed zenith angle. Observations that indicate saturation or other total power anomalies are rejected. Heavy cloud cover or precipitation often causes large fluctuations in total power. Such periods are identified by inspection of the total power time series and manually discarded. In figure 3.7, the average P for flux procedures executed during a thunderstorm is shown, illustrating the sort of behavior that is selected for removal. These manual flags are implemented as date interval flags as described above.

3.2.1.5 Measured Uncertainty Cuts

We reject flux density measurements with anomalously large measured uncertainties, σ_{15}, defined by equation (2.44). However, a straightforward cut at a fixed value or a fixed multiple of the expected thermal uncertainty introduces a bias against larger flux densities. This occurs because there are contributions to the measured error that are proportional to the flux density of the target radio source, such as telescope tracking errors. We therefore apply a flux density-dependent threshold and discard flux density measurements for which

$$\sigma_{15} > \zeta\sqrt{1 + (\rho \cdot S_{15})^2}. \tag{3.1}$$

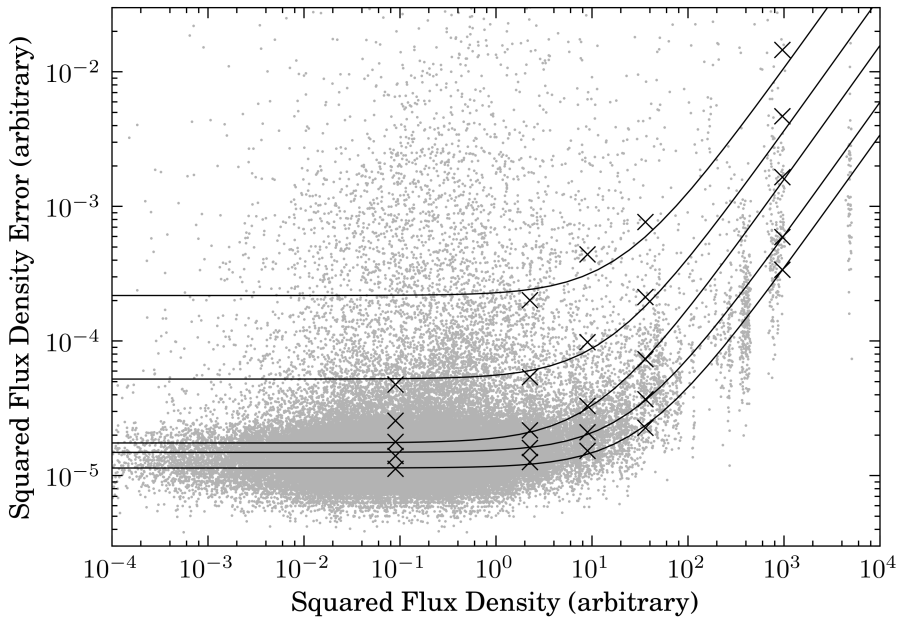

Figure 3.8. Logarithmic plot of σ_{15}^2 versus S_{15}^2 for the data used to fit ρ in the measured uncertainty filter. Grey dots are the individual data, black crosses are the 25th, 50th, 75th, 90th, and 95th percentile values in the bins indicated in table 3.1, and the solid lines are linear fits to the black crosses for each percentile. The flux density values are plotted in approximate Jy units.

Table 3.1. Flux density bins used for fitting the measured uncertainty filter parameters

Bin (Jy)	Center (Jy)	# Data	Threshold (Jy)	Percent Rejected
0.2–0.4	0.3	19498	0.0208	98.4
1–2	1.5	6771	0.0217	97.1
2–4	3.0	2574	0.0242	96.3
4–8	6.0	1230	0.0324	95.9
22–40	31	230	0130	97.0

Note: The threshold is computed at the center of each bin for this study.

To determine the optimal values of ζ and ρ, we examined the data collected between March and October, 2009. In figure 3.8, we plot the square of the measured uncertainty for each flux density as a function of the square of that flux density in grey. The black crosses indicate the 25th, 50th, 75th, 90th, and 95th percentile of the data in each of the five flux density bins listed in table 3.1. The solid lines show linear fits to the data. Although there is peculiar behavior at low flux densities, the high-flux density behavior is similar in all the fits. We used the 50th percentile (i.e., median) fit to determine the optimal value $\rho = 0.200$. Finally, $\zeta = 0.0208$ Jy was set to discard as many obviously bad flux density measurements as possible while maintaining selectivity. In table 3.1, we tabulate the percentage of data rejected in each bin, which is reasonably insensitive to the flux density of the bin. Overall, about 2% of the data are eliminated by this filter.

3.2.1.6 Switched Difference Cuts

We also use the switched difference μ, defined by equation (2.46), to determine whether flux density measurements might be contaminated by systematic errors. The expected value of μ is 0, provided that the ground spillover and atmospheric noise in the *ant* and *ref* beams are identical. Pointing and tracking errors again give flux density-dependent contributions to μ, so to avoid bias against brighter radio sources we flag points where

$$\left|\frac{\mu}{\sigma_{15}}\right| > \beta \cdot \frac{(\mu_0 + \rho_s \cdot S_{15})}{\sqrt{1 + (\rho_t \cdot S_{15})^2}}. \tag{3.2}$$

The optimum values of the parameters ($\beta = 5$, $\mu_0 = 1.148$, $\rho_s = 0.0682$, and $\rho_t = 0.0243$) are determined from the data. Figure 3.9 shows the data that were used to determine the parameters. The switched difference values for FLUX procedures between October 2008 and October 2009 are plotted in grey. These were binned into eight bins (in Jy, 0–0.1, 0.1–0.3, 0.3–1, 1–3, 3–10, 10–25, 25–40, 60–80; there were no data between 40 and 60 Jy) and the median value in each bin was computed; these are shown by the black points. The parameters in equation (3.2) were fitted to the binned data with $\beta = 1$, giving the other parameters. The value $\beta = 5$ was chosen to drop as many suspect data points as possible while maintaining the selectivity of the filter.

This filter discards about 2% of flux densities. Because the tracking performance of the telescope should not change, this procedure is expected to give consistent results across epochs for a fixed set of parameters. This expectation appears to be correct—approximately equal fractions of the data are dropped from each epoch. Among bright ($S_{15} \geq 10$ Jy) sources in the 2008–2009 period, 1.4% of data were flagged by this filter. This is comparable to the 1.7% overall flagging rate for the same period, which indicates that we have successfully removed the bias against bright flux densities.

3.2.2 Flux Density Calibration

The data recorded by the control system is measured in the arbitrary units that come from the 16-bit analog-to-digital converter, referred to as DU. In this section, we describe the procedure for determining the calibration factor to convert DU into physical units. This calibration factor changes with time due to, e.g., fluctuations in the receiver gain, intentional changes in the programmable attenuator setting, or inadvertent but unavoidable changes in the loss or mismatch in the signal path between the antenna feed and the receiver when components are disassembled and reassembled for maintenance. Our fundamental calibration strategy is to use the stable CAL diode output to account for short-term variations within the receiver, then periodically determining the ratio between the CAL diode level and an astronomical calibrator (3C 286) to account for longer-term variations, including those outside the receiver.

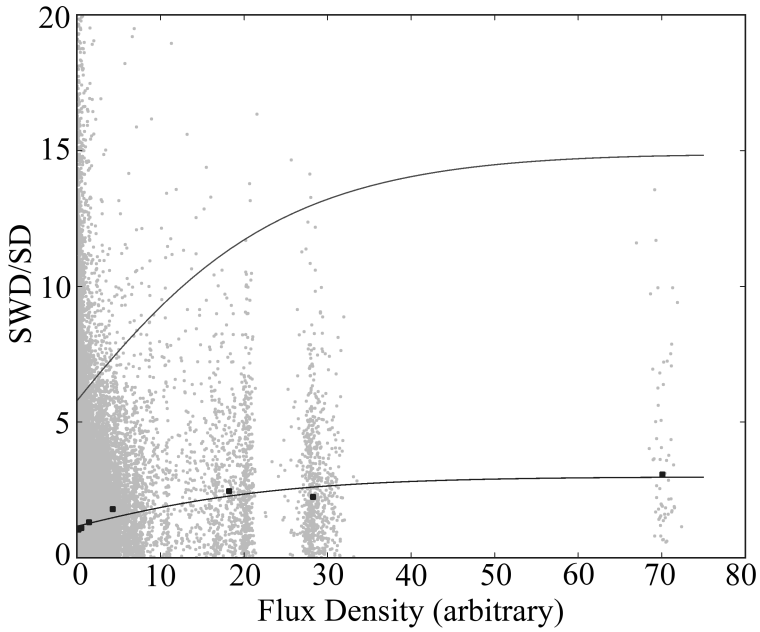

Figure 3.9. Plot of the switched difference, μ normalized by σ_{15}, versus S_{15} for FLUX procedures between October 2008 and October 2009. Grey points are the individual data points, black points are the binned medians. The lower black line is a fit of equation (3.2) to the black data points with $\beta = 1$. The upper black line corresponds to the same parameters except with $\beta = 5$, corresponding to the adopted threshold. The flux density values are plotted in approximate Jy units.

3.2.2.1 Relative Calibration

To correct for slow gain fluctuations of the receiver, we first divide each flux density measurement by a calibration factor measured using the small noise diode CAL. A measurement of the strength of the CAL diode is made after each pointing observation, and no less than once per hour. Gain fluctuations that affect the Dicke-switched data are rather slow, so the calibration factor is averaged over a two-hour window, centered on the time of the flux density measurement. If there are fewer than two good measurements of the strength of the CAL diode in that window or if the measurements disagree by more than 10%, then the flux density observation is discarded.

Due to gravitational deformation of the telescope structure and the increase in atmospheric attenuation with airmass, the effective antenna gain varies substantially with zenith angle. We model this variation with a polynomial gain curve and scale flux density measurements to remove the effect, as described in section 2.1.2.3. Day-to-day changes in atmospheric opacity are found to vary with $<1\%$ rms, so these variations are accounted for within our error model and no correction is applied.

Additionally, the optimal axial focus position varies with zenith angle, as well as solar zenith angle and elongation. During observations, the focus position is set using a polynomial model of the zenith angle variation, and a correction is applied during calibration using a more complete model that accounts for solar zenith angle and elongation. This procedure is described in more detail in section 2.1.2.3.

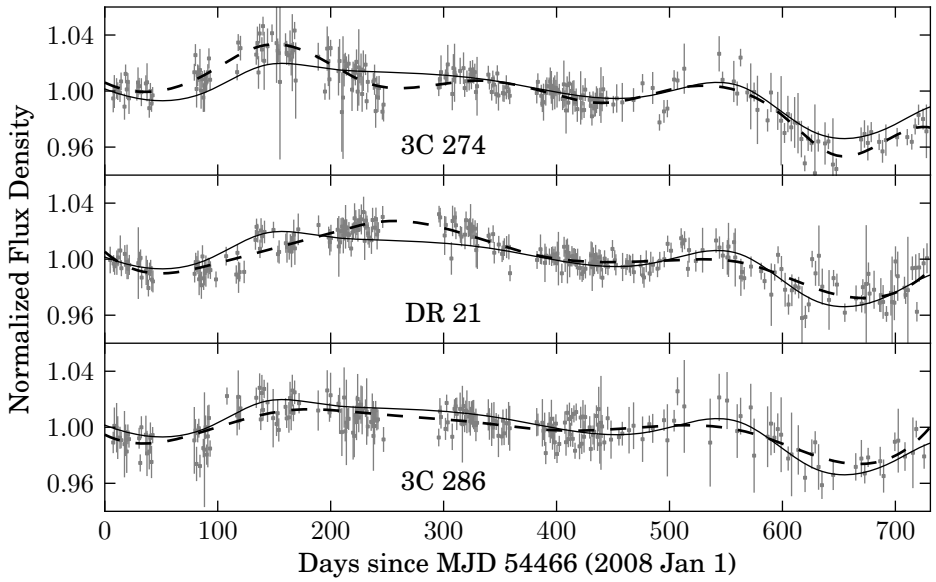

Figure 3.10. Normalized flux densities for 3C 274 (top), DR 21 (center), and 3C 286 (bottom) after outlier removal. Each light curve is normalized by its median. The solid line in each plot is the spline fit to the combined data. The dashed line is a similar spline fit to the data for the individual source.

The combined effect of these corrections is a factor, κ_{rel}, that is computed for each flux density measurement. This is one component of the overall κ introduced in equation (2.43).

3.2.2.2 Long-Term Trends in 3C 286, 3C 274, and DR 21

After carrying out the above editing and calibration steps we returned to the residual 1%–2% long-term (about 6-month) variations in the light curves for stable-flux-density calibration sources. We chose 3C 286, 3C 274, and DR 21 for this study because they are well-known to be stable on timescales of many years. The fractional variations in flux density of these objects are shown in figure 3.10 and are clearly correlated, indicating the presence of an unidentified source of multiplicative systematic error. For each of these sources, we removed 2σ outliers in a 100-day sliding window and normalized the resulting data by the median flux density. We then combined the data for all three sources and fitted a cubic spline to the result.

We apply the corresponding correction to all light curves in our program by dividing each flux density by the value of this spline. Figure 3.11 shows the residuals for the three fitted sources after dividing out the spline fit. The 1% residual variation that remains is the level of systematic uncertainty after correction for this long-term trend.

Each time we perform a reduction, it is necessary to check whether these calibration splines need to be updated. This is a bit challenging because it can be difficult to determine whether a few points above or below the mean flux density for a stable source merely represent a few statistical deviations or actually indicate a new trend. We normally assume the former, only extending the spline fits when there is clear evidence for a correlated trend among the calibrator sources. We extend the spline curves by refitting a cubic spline to

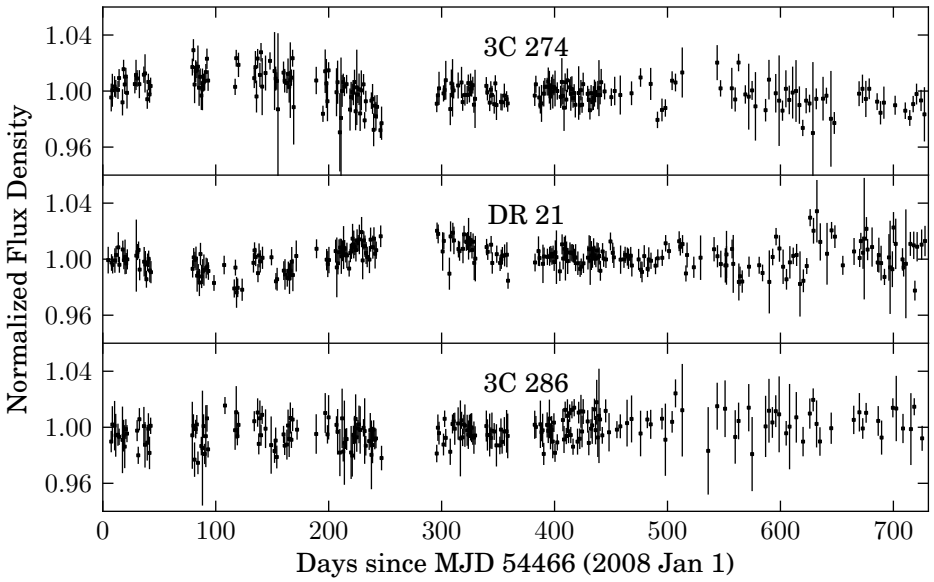

Figure 3.11. Normalized flux densities for 3C 274 (top), DR 21 (center), and 3C 286 (bottom) after dividing by the spline fit to remove long-term systematic trends.

Table 3.2. Calibrator spline epochs

Epoch	Start		End	
	MJD	Date	MJD	Date
1	54466	01 Jan 2008	55197	01 Jan 2010
2	55197	01 Jan 2010	55624	04 Mar 2011
3	55624	04 Mar 2011	55728	16 Jun 2011

the entire light curve for the calibrators, adjusting the number of knots in the spline to match the end of the previously determined spline as well as possible. To avoid changing the values of old data, however, we continue to use the old spline for the older data. The boundaries between spline fits are stored using a set of *calibrator spline epochs* that are updated as necessary. These epochs are tabulated in table 3.2.

3.2.2.3 Flux Density Calibration

We divide our observation period into *flux density calibration epochs* characterized by a consistent ratio between the calibration diode and feed horn inputs to the receiver. This ratio might change if, for example, the signal path is disconnected and reconnected for maintenance, resulting in a slight change in loss along one path. Within a single epoch, the ratio of the CAL diode signal to a stable astronomical source should therefore be constant. Table 3.3 lists the epochs we have used in our analysis. Flux density calibration[9] is applied to each epoch separately.

[9]N.B., the script that implements the flux density calibration steps is named the "absolute calibration" script, although we do not actually perform absolute calibration.

Table 3.3. Flux density calibration epochs

Epoch	Start		End	
	MJD	Date	MJD	Date
1	54466	01 Jan 2008	54753	14 Oct 2008
2	54753	14 Oct 2008	54762	23 Oct 2008
3	54762	23 Oct 2008	55420	12 Aug 2010
4	55420	12 Aug 2010	55482	13 Oct 2010
5	55482	13 Oct 2010	55511	11 Nov 2010
6	55511	11 Nov 2010	55543	13 Dec 2010
7	55543	13 Dec 2010	—	—

For each epoch, a calibration factor is determined from regular observations of the primary calibrator, 3C 286. We adopt the spectral model and coefficients from Baars et al. (1977). At our 15 GHz center frequency, this yields 3.44 Jy, with a quoted absolute uncertainty of about 5%. The calibration factor for epoch i, κ_i, is the ratio of the adopted flux density for the calibrator to the weighted mean of the observations:

$$\kappa_i = \frac{3.44 \text{ Jy}}{\left(\sum S'_{15} \cdot {\sigma'_{15}}^{-2}\right) / \left(\sum {\sigma'_{15}}^{-2}\right)}, \tag{3.3}$$

where S'_{15} and σ'_{15} denote the flux densities for the calibrator with only the relative calibration applied.

The total calibration factor for a flux density measurement in equation (2.43) is then $\kappa = \kappa_{rel} \cdot \kappa_i$, and reflects both relative and flux density calibration. Comparing our calibrated flux densities for 3C 48, 3C 161, and DR 21 with the Baars et al. (1977) values, we find a scale error of $(-0.8 \pm 4.1)\%$. This is probably a conservative estimate (i.e., an overestimate) of the scale error because some of this disagreement may result from variation in the sources. After correcting for different flux density scales, cross-checks of our calibration against 14.6 GHz observations of a number of common sources observed with the Effelsberg 100 m telescope through the F-GAMMA project confirm the overall accuracy of our flux density scale.

3.2.3 Uncertainties in Individual Flux Density Measurements

In a perfect observing system with no sources of systematic error the uncertainties in the flux density measurements would be given by the thermal noise on each observation. In practice there are many sources of systematic error, including the effects of weather and the atmosphere, mispointing due to wind, and focus errors. Many of these are correctly identified and accounted for in the automatic and manual editing and calibration described in the preceding sections. However, even after flux density measurements affected by these problems are filtered out, there remain many observations that are significantly affected by systematic errors. Such systematic errors can lead to significant errors in the measurement that are not reflected in the thermal noise of the observation and can give rise to bad flux density measurements with small thermal errors. This leads to *outliers* in the light curves, i.e., points which do not lie close to the level determined from interpolation of adjacent observations and which have small errors. The task of identifying and eliminating or

Table 3.4. Error model parameters

Parameter	Pointing Calibrator	Normal Source	
		Early	Late
ϵ	0.0057	0.0200	0.0135
η	3.173	3.173	3.173

Note: The "early" and "late" periods are before and after 16 March 2009 (MJD 54906), respectively.

allowing for the wide variety of systematic errors leading to such outliers is challenging and time-consuming. Great care must be taken not to assume that the behavior of the source is known, and hence to eliminate a real and potentially extremely interesting flux density variation.

3.2.3.1 Error Model

We first apply an error model to determine the uncertainty of each flux density measurement:

$$\sigma^2_{\rm total} = \sigma^2_{15} + (\epsilon \cdot S_{15})^2 + (\eta \cdot \psi)^2 \ , \tag{3.4}$$

which is an extension of the model described in Angelakis et al. (2009). The first term represents the measured scatter during the flux density measurement. This includes thermal noise, rapid atmospheric fluctuations, and other random errors. The second term adds an uncertainty proportional to the flux density of the source. This term allows for pointing and tracking errors, variations in atmospheric opacity, and other effects that have a multiplicative effect on the measured flux density. In the third term ψ is the switched power, defined by equation (2.45). This term takes account of systematic effects that cause the A-B segment of the flux density measurement to differ from the C-D segment, such as a pointing offset between the A and D segments, or some rapidly varying weather conditions.

The error model is defined by the two parameters, ϵ and η, whose values must be determined from the observations. Because ϵ describes the error contribution due to pointing errors, its value depends on whether a source is used as a pointing calibrator. Furthermore, for non-pointing calibrator sources, ϵ is found to differ between the scheduling algorithms used before and after 16 March 2009 (MJD 54906). The parameter, η, is found to be adequately described by a single value for all sources and all epochs. The adopted values are given in table 3.4.

For pointing calibrator sources, both ϵ and η were estimated simultaneously using the stable flux calibrators 3C 286, 3C 48, 3C 161, and DR 21. Due to systematic errors, these sources and other stable-flux density calibrators show long-term variations of 1%–2% so we fitted a 7th-order polynomial to remove this trend from each source, then computed the residual standard deviation, median flux density, the rms, and mean ψ for each source, then used these to fit the error model parameters.

To determine the error model parameter ϵ for ordinary sources, we selected 100 sources that exhibited little variation or slow, low-amplitude variations in flux density, between the start of our program and 05 Au-

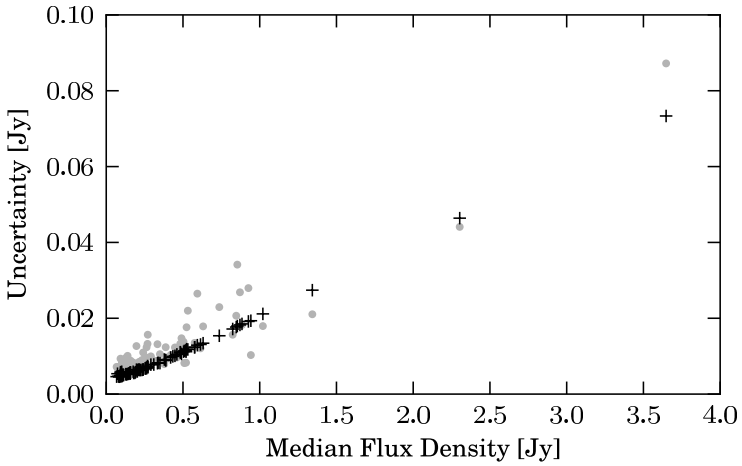

Figure 3.12. Residual standard deviation (grey points) and fitted ϵ-only error model values (black crosses) for ordinary sources in the early (MJD $<$ 54906) period. The fit in the late period is similar. A single high-flux density data point was omitted to limit the scale.

gust 2009 (MJD 55048). This interval was split into two periods, "early" and "late," at MJD 54906 and this procedure was separately applied to each period. For each light curve, we fitted and removed a second-order polynomial trend, then iteratively removed outlier data points with residuals greater than three standard deviations. We repeated the fitting and outlier removal until no further outliers were removed and we discarded any source with fewer than 10 remaining data points (retaining 94 and 88 sources in the early and late periods, respectively). From the surviving points in each light curve, we computed the median and the rms flux densities, and the standard deviation of the residuals. We then fitted equation (3.4) to these data, omitting the η term. The data and the error model results for the early period are shown in figure 3.12. We then adopted the same value of η for these sources as was determined for pointing calibrator sources.

3.2.3.2 Scaling of the Nonthermal Error

The uncertainty model we have introduced combines two qualitatively different components. The first component is that directly obtained during the flux measurement, σ_{15}, which represents random errors such as thermal noise and rapid atmospheric fluctuations. The second, quantified by the ϵ and η parameters, is introduced to take into account other, flux-density–dependent effects. Thus far, we have assumed these to be source independent. However, many sources exhibit coherent long-term variations with random scatter about those that is clearly smaller than what would be expected as a result of the quoted errors. This indicates that the assumption of source-independent ϵ and η is invalid and has resulted in an overestimation of the actual uncertainty in some cases. We now describe a correction that is applied to the errors to reduce this effect. The method for computing correction factors described here was developed and implemented by Walter Max-Moerbeck based on ideas and requirements that we developed together.

To correct these constant scale factors on a source-by-source basis, we begin by fitting a cubic spline to the light curve, adjusting the number of knots to achieve a residual χ^2 per degree of freedom near to one. Due to the large number of sources and the requirement of an uniform and consistent method, an automated method was developed for this procedure. We begin by removing any extreme outliers using a low-order cubic spline fit, rejecting the data in the top 5% of the absolute residuals. Next, for each number N_k in a reasonable range (typically 1 to 80), we fit a cubic spline with N_k knots to the remaining data.[10] Not all the fits are acceptable—some cases have obviously correlated residuals or a large departure from normality. Acceptable fits are selected using two statistical tests: Lilliefors' test for normality (Lilliefors 1967) and the runs test for randomness (e.g., Wall & Jenkins 2003).[11] Only the fits for which both null hypotheses cannot be rejected at the 10^{-3} level are considered acceptable. For each acceptable fit, a scale factor that makes the χ^2 per degree of freedom equal to one is calculated. Among the scale factors for all the acceptable fits, the median scale factor is selected as the final correction. The value of the scale factor is not very sensitive to the exact number of polynomial sections. A typical example of the behavior of the scale factor is shown in figure 3.13.

Using the correction factor, we rescale only the nonthermal part of the errors (the S_{15} and ψ terms in equation (3.4)). This correction is only applied to those sources for which the resulting correction factor is smaller than one (i.e., the rescaling would result in *smaller* errors). The latter choice was made for two reasons. First, a correction factor larger than one simply indicates that the spline fit cannot provide an adequate description of the data. This may result from a light curve more variable than can be fit by spline with a given number of knots, so such a correction could mask real variability. Only the reverse is cause for concern—when the spline fit is too good a fit, given the quoted errors. Second, this choice ensures a smooth transition between scaled and nonscaled errors, as the transition point (correction factor equal to one) is equivalent to no error scaling.

The error scale factors were first computed only for the CGRaBS sample using the two-year data set, then were computed for all sources using the 42-month data set. For the CGRaBS sources, the error scale factors changed somewhat between the two data sets. In figure 3.14, the histogram of the differences between the 42-month error scale factor and the two-year error scale factor for each source is plotted for the sources that had a scale factor less than one (i.e., had a correction applied) in either interval. The mean change is an increase by 0.134.

[10] We use the MATLAB Spline Toolbox function spap2, which automatically selects the positions of the knots for the spline.

[11] We have used the implementation of both tests that are part of the MATLAB Statistics Toolbox.

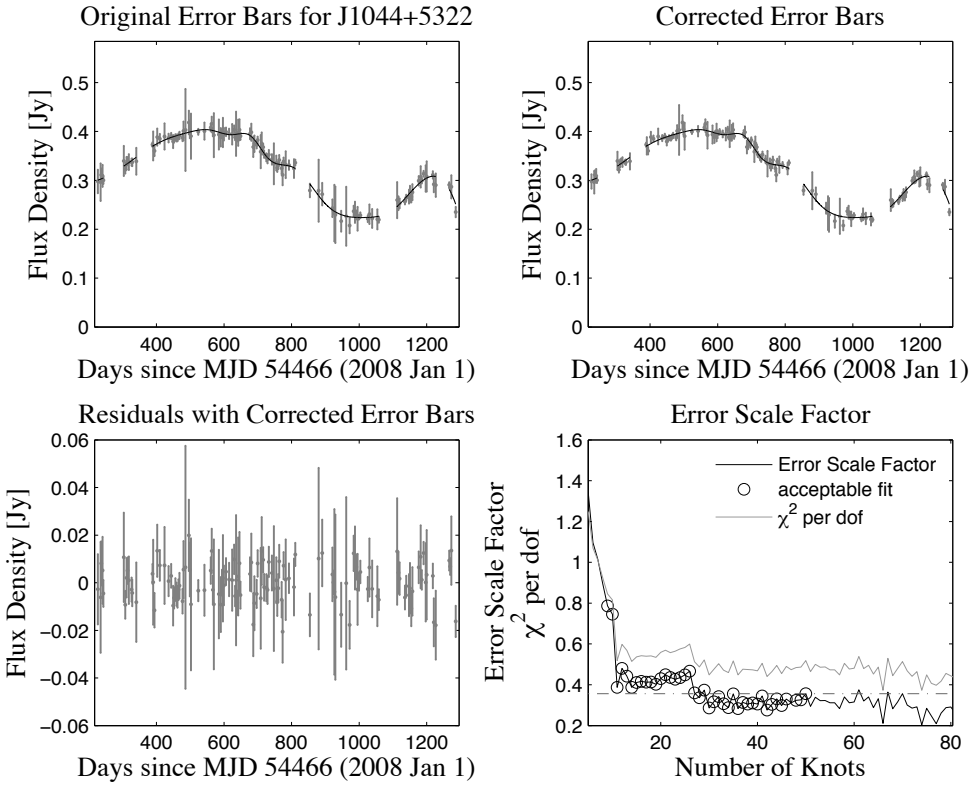

Figure 3.13. Example of the error scale factor correction using data for J1044+5322. The two upper panels show the light curve with the original (left) and corrected (right) error bars (grey points) and a typical spline fit (black line). The bottom left panel shows the residuals from the spline fit using the corrected error bars. In the bottom right panel, the χ^2 per degrees of freedom (solid grey line) and correction factor (solid black line) are shown, with black circles marking the correction factors for fits that pass the acceptance tests, and a dashed line showing the adopted correction factor for the source (0.356).

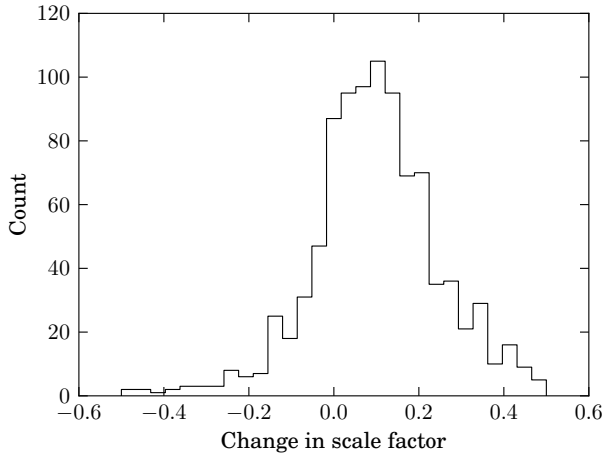

Figure 3.14. Histogram of the change in error scale factors for CGRaBS computed using the 42-month versus the two-month data sets. Only sources for which one or both error scale factors were less than one are included.

3.3 Storing and Retrieving the Data

We now turn our attention to the method by which the data for the program are stored. This is accomplished using a MySQL[12] database. Two such databases are used by this program: the *reduction* database, which is used by the data reduction pipeline, and the *results* database, which is used to store the fully reduced data. In this section, we will primarily discuss the reduction database. We begin this section by discussing the design principles that guided the design of the database schema, then describe the most important components of the schema. Finally, we will briefly describe the results database.

In this section, we present diagrams of several of the major parts of the reduction database. Diagrams of the remaining tables in the database are found in appendix B. Tables listing the contents of the domain tables are also found in the appendix.

3.3.1 Database Design Principles

A schema is, informally, the set of constraints that define the tables that store the data within the database. MySQL implements a relational database (at least approximately—there are some technical differences between an SQL database and a true relational database), so each table within the database represents a relation. That is, each table represents true values of some true-or-false statement, called a predicate. For example, our `Source` table is defined by the predicate, "An astronomical source with right ascension α, declination δ, redshift z, etc., is part of the monitoring program." The schema, then, is a technical definition of the predicates that define the tables in the database. The relational database was a groundbreaking development—it represented the first database to be rigorously defined and analyzed in mathematical terms. However, in this section, we will restrict our discussion to informal terms, and in some cases describe behavior that is specific to MySQL. In most cases, other SQL databases will have similar behavior.

3.3.1.1 Normal Forms

Although there is great freedom to design database schema, a well-known set of conventions, or *normal forms*, has been developed. By organizing a schema according to the normal forms, redundancies and inefficiencies in the database structure are avoided. We have consciously attempted to apply the first three normal forms to the reduction database schema.

The *first normal form*, or 1NF, requires that the columns in a table not contain any repeating groups. For example, because astronomical sources are frequently known by many names or numbers according to different catalogs, it is tempting to define a table of sources with columns, "Name 1," "Name 2," etc. This violates 1NF because each of these columns represents the same sort of information. To comply with this, we must restrict ourselves to nonrepeating data columns. We could comply with this normal form in a source table by storing multiple copies of each source row, one for each name.

[12] `http://www.mysql.com/`

This is obviously inefficient—the data for each source is repeated many times, both requiring extra space and extra effort if any of those data need to be updated. The *second normal form*, or 2NF, addresses this. It requires that rows within a table not contain this sort of redundant information. Thus, rather than repeating the data for each source for each of its names, we must instead create a separate table of source names. Each entry in this table should contain a single source name and a pointer to the row in the source table to which that name applies. The pointer is implemented using a *key* (or *candidate key*) in the source table—a key is a subset of the data in the row that uniquely identifies that row in the table. Each table must have one key defined to be the *primary key*.

Although in principle a key can be constructed from the actual data in the table, in our database, we use the convention of assigning each row a unique integer identifier known as a *surrogate key*. These keys are identified by a "_ID" suffix in the column name. In our example, the reference between the source name table and the source table would then be implemented by storing the surrogate key from the source table in source name with each name for that source. This reference to a key in another table is called a *foreign key*.

The *third normal form*, or 3NF, requires that all fields within a row must have a functional dependence on the key for that row. That is, all the data within the row must actually be tied to the entity described by that row. In section 3.3.2 below, we give an example of the application of 3NF. Several other normal forms exist, but a full discussion is beyond the scope of this section. We have not explicitly attempted to normal forms beyond 3NF to our database schema. In a few cases, we have consciously broken even the first normal form. This has been done to simplify the database schema. For a database of modest size, like ours, this is generally acceptable.

3.3.1.2 Table Indexes

Searching for an entry in an unsorted table is an $\mathcal{O}(N)$ operation—a table with N entries requires time proportional to N. When executing a query that joins several large tables in the database, the search space is the Cartesian product of those tables, with a total number of rows equal to the product of the number in each table. As a result, N can easily grow to be astronomically large, resulting in extremely slow response from the database. Modern databases like MySQL attempt to organize queries to avoid such enormous searches, but in many cases the database designer must provide additional assistance by defining *indexes* on the tables.

An index is a precomputed ordering of the rows in a table using the data values in that row. In MySQL, searching a table using an index is often a constant-time ($\mathcal{O}(1)$) operation, compared to the linear-time ($\mathcal{O}(N)$) full-table search. The response time to queries can be improved enormously by appropriately defining indexes on large tables. Adding an index to a table does incur some costs, however. While an index speeds up data retrieval, it requires that the index be recomputed whenever data are added to the table. Thus inserting data into the database is generally slowed by each index. Additionally, each index increases the disk space used by the database table.

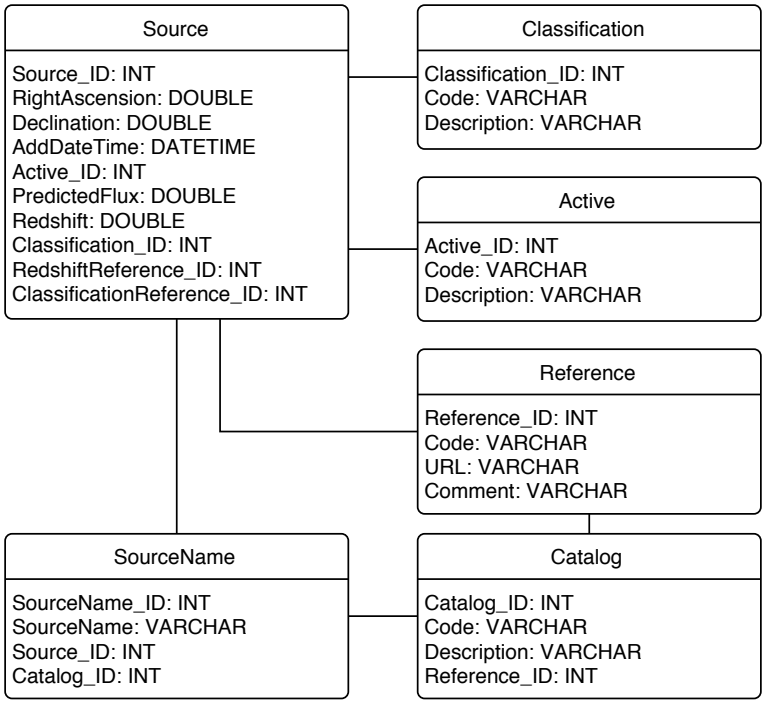

Figure 3.15. Diagram illustrating the database tables used for storing source data.

An index is defined by an ordered list of columns from the table. Indexes can be defined as *unique*, which will prevent more than one row with the same value in each index column from being added to the table. Every table has at least one unique index—the primary key. In the reduction and results databases, we have used indexes both to improve the speed of data retrieval and to enforce uniqueness constraints, for example to prevent accidentally inserting multiple copies of a FLUX result.

3.3.2 Representing Sources

In figure 3.15, a diagram illustrating the database tables relevant to storing source names, coordinates, and other data is shown. Each box in the diagram represents a table in the database with the name given in the header. The entries below the header give the names and data types of the columns in that table. The first column listed for each table is the surrogate primary key for that table. Lines between boxes indicate a foreign key reference between the two tables, which can be identified by a column in the referring table with the same name as the primary key of the table to which it refers.

Rows in the `Source` table represent astronomical sources. For each source, the J2000 right ascension and declination are stored in degrees. The `PredictedFlux` column contains the recent typical 15 GHz flux density measured for the source, which is used by the scheduler to choose suitably bright pointing calibrator sources. The redshift and optical classifications are also stored, as is the date at which the source was added to the database.

In keeping with the 1NF and 2NF rules described above, names for the sources are stored in a separate `SourceName` table which associates one or more names with each source. The `Catalog` table defines the names of catalogs of sources, and a unique index prevents more than one source name within a single catalog to be created (although different catalogs can use overlapping source names).

The `Active` and `Classification` tables (and also the `Catalog` and `Reference` tables) are examples of *domain tables*. These define the domain of values that are valid for entries in a column in another table. In this case, the `Active` table defines options for specifying whether a source is actively being observed as part of the monitoring program, and the `Classification` table defines the range of classifications that can be assigned to a source. Use of a domain table rather than, say, a free-form `VARCHAR` string field, ensures that entries in a table use a consistent set of values.

The `Reference` table stores a list of literature references that are used to track references for the source catalogs as well as for the redshift and classifications stored for each source. The `Reference` table stores a short name for the reference, plus a comment and a full reference to the cited material (the `URL` column). This is an example of applying 3NF: we could have opted to store both the short name for the reference and the full reference with each redshift in the `Source` table. However, including the full reference would violate 3NF because that reference is a property of the cited paper, not of the astronomical source. By separating these data into the Reference table, we are able to update the `URL` field for all the references simply by changing one value in one row of the `Reference` table rather than replacing every instance in the `Source` table. In addition, we can reuse the `Reference` table entries to provide values for the source classifications in the `Source` table and for the `Catalog` table.

There is a minor violation of 2NF in the `Source` table: the `ClassificationReference` and `RedshiftReference` columns are technically repeated instances of the same type of information. These should properly be stored through an additional table linking `Source` table entries to the `Reference` table with a reference type (“redshift” or “classification”) for each link. However, we decided that the overhead of adding this additional table was not warranted for the minor benefit of strict 2NF compliance.

3.3.3 Storing Observation Data

In figure 3.16, a diagram showing the tables relevant to storing observation data from the monitoring program is shown. The `Reduction` table has foreign key references to tables not shown here. It is discussed further in section 3.3.4.

3.3.3.1 FLUX and POINT Procedures

Two tables, `PointProcedure` and partially calibrated `FluxProcedure` store the POINT and FLUX procedures that result from the low-level reduction script. The data stored in these tables are largely based on the data that were reported for each procedure by the VAX control system. Some additional data, such as the

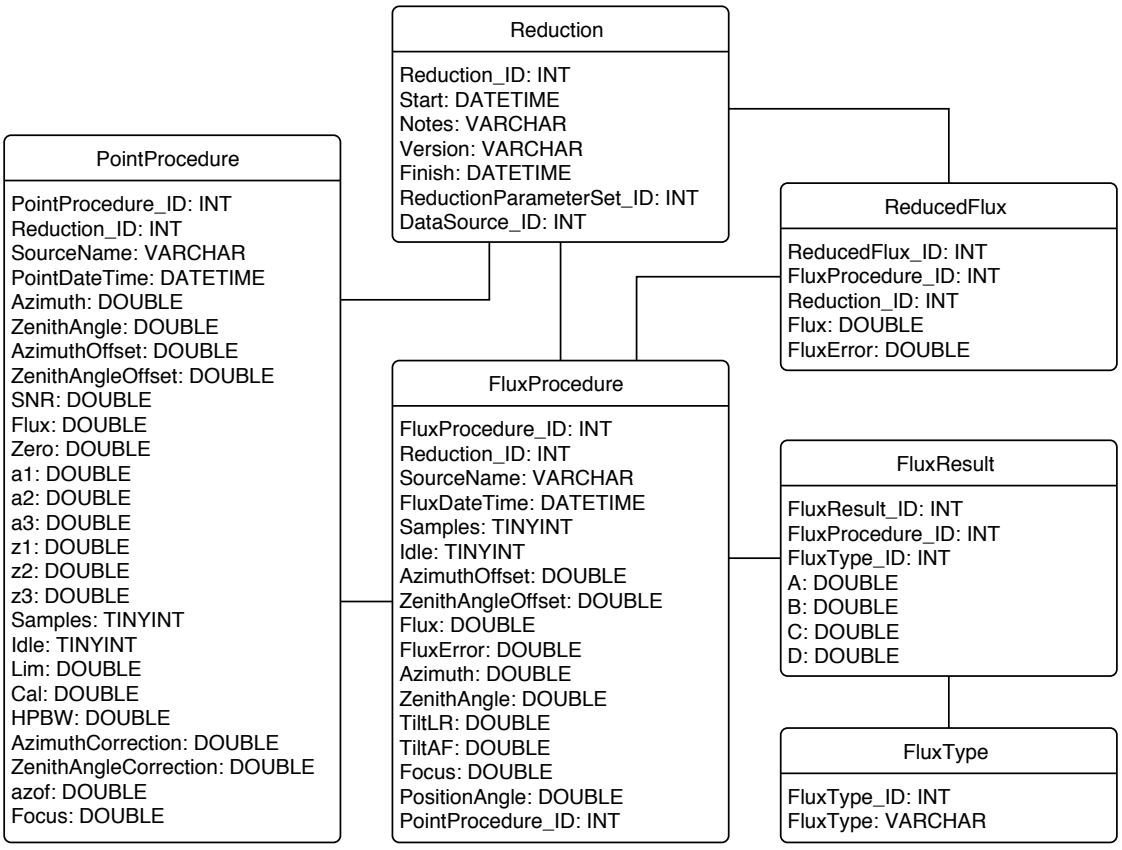

Figure 3.16. Diagram illustrating the database tables used for storing observation data.

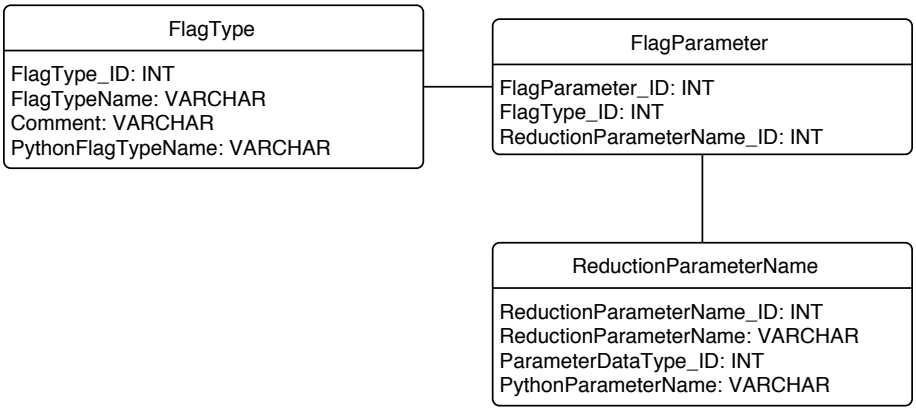

Figure 3.17. Diagram illustrating the database tables used for storing flag types and parameter definitions.

tilt meter readings and focus position, are stored in these tables for convenience. Each `FluxProcedure` entry refers to a `PointProcedure` entry to indicate which POINT procedure was used to compute the pointing offset in effect during the FLUX. For FLUX procedures executed with no measured pointing offset, this reference is null.

The `SourceName` entry in these tables contains whatever source name appeared in the log file or archive as the name of the target for that observation procedure. It is *not* a reference to a source name in the `SourceName` field. Resolving which program source is referred to by this name is part of the flux density calibration script.

Each FLUX procedure reports two sets of integrations for the A, B, C, and D segments. One is the switched average for each segment, the other is the total power. In keeping with 1NF and 2NF, these are stored as separate entries in the `FluxResult` table to avoid repeated encoding of the same type of data in the `FluxProcedure` table.

The `ReducedFlux` table stores the results of the high-level calibration script for each FLUX. Because there is a one-to-one correspondence between `FluxProcedure` and `ReducedFlux` rows, the latter refers back to the former rather than than store a duplicate of the the data from that table. Only the `Flux` and `FluxError` values need to be stored (and the `Reduction_ID` field will refer to a different `Reduction`).

3.3.3.2 Flag Tables

Diagrams of the tables that contain the data editing flag types and their parameters are shown in figure 3.17. The parameter values are actually stored in the `ReductionParameter` table described in section 3.3.4. The `FlagParameter` table connects each flag with the correct `ReductionParameterName`.

Another set of tables are used to store the date intervals to be flagged. Diagrams of these tables are shown in figure 3.18. Each entry in the `FlaggedDate` table describes one interval to be flagged. A comment is stored with each entry for logging purposes, and each is assigned a `FlaggedDateType` that classifies the

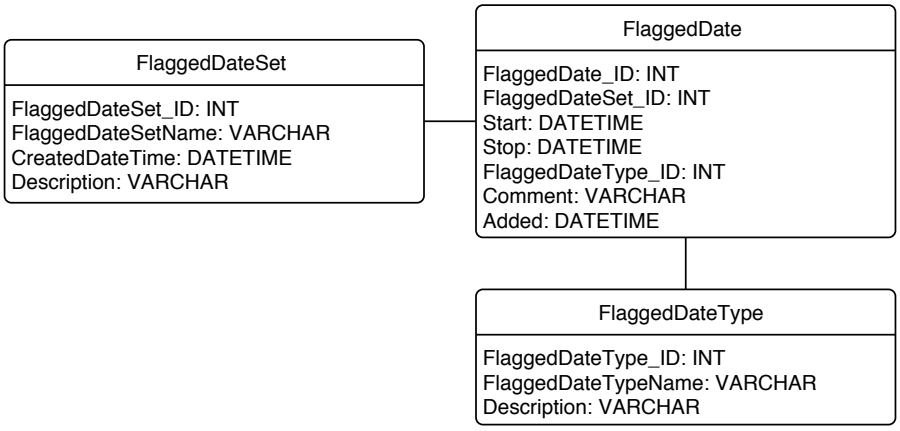

Figure 3.18. Diagram illustrating the database tables used for storing date interval flags.

reason for the flagging. The `FlaggedDateSet` table is used to define independent sets of `FlaggedDate` entries so that separate reductions can be stored in the database simultaneously without requiring them to share common date interval flags.

3.3.4 Managing Data Reduction

In figure 3.19, a diagram of the tables used to manage the data reduction process is shown. The `Reduction` table stores a record of every execution of a low-level or high-level reduction script. Flux density calibrations are not stored because these do not modify the data in the reduction database. When a reduction is begun, a new entry in the `Reduction` database is created with the `Start` field filled in with the UTC date and time of creation. Upon completion of the reduction process, the `Finish` field is filled in.

The low-level reduction script reduces data in units of one day. Upon completion of a low-level reduction, the dates that were processed are noted in the `ReducedData` table. Each date is associated with the `Reduction` during which it was processed, and the type of reduction performed (i.e., whether it was processed using the VAX control system scripts, the MCS control system scripts, or was copied from an existing low-level reduction) is noted using the `ReductionType` table. Using these tables, the reduction scripts can automatically determine what time periods need to be reduced.

Each `Reduction` is given a version and the name of a set of reduction parameters. The version is stored as a free-form string in the `Version` field of the table. There is no domain table for this, the user must keep track of the version names outside of the database. By convention, low-level reductions are simply numbered, while high-level reductions are given numbers with the prefix “HL” (e.g., “HL15”). The data reduced for this thesis were given the special version “JLR01” which is based on low-level data from version 15.

The parameter list is stored as a reference to the `ReductionParameterSet` table. Entries in the `ReductionParameter` table define the set of parameters and their values that belong to each set. The names and data types for the reduction parameters are specified by the `ReductionParameterName` and

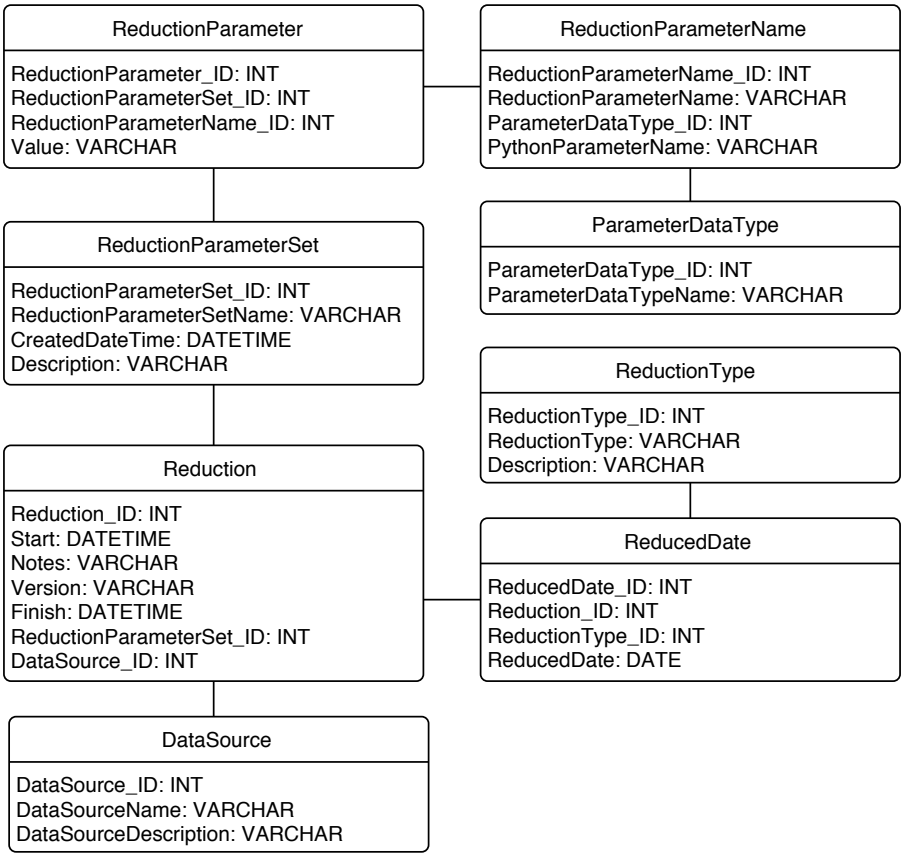

Figure 3.19. Diagram illustrating the database tables used for storing reduction data.

`ReductionParameterType` tables. Within the database, reduction parameter values of all types are stored as `VARCHAR` strings. When a Python script reads these from the database, the value is parsed according to its `ReductionParameterType` value. Multivalued parameters are, by convention, stored as comma-separated string parameters.

3.3.5 The Results Database

As of the time of writing, our 15 GHz light curve data for the CGRaBS sample are made publicly available through the web within a few months of observation.[13] This web page is powered by another MySQL database, the *results database*, into which our fully reduced data are inserted. This database schema, scripts to create and populate it, and the HTML and PHP code for the web interface were developed by Matthew A. Stevenson. Starting from his work, I have made a number of small changes to the scripts and database schema and added indexes to the database to improve performance. The source table schema described in section 3.3.2 is based on the design from the results database.

[13] `http://www.astro.caltech.edu/ovroblazars`