PUBLICATIONS OF THE ASTRONOMICAL SOCIETY OF THE PACIFIC, **126**:674–710, 2014 July

IPAC Image Processing and Data Archiving for the Palomar Transient Factory

RUSS R. LAHER,[1] JASON SURACE,[1] CARL J. GRILLMAIR,[1] ERAN O. OFEK,[2] DAVID LEVITAN,[3] BRANIMIR SESAR,[3] JULIAN C. VAN EYKEN,[4] NICHOLAS M. LAW,[5] GEORGE HELOU,[6] NOUHAD HAMAM,[6] FRANK J. MASCI,[6] SEAN MATTINGLY,[7] ED JACKSON,[1] EUGEAN HACOPEANS,[8] WEI MI,[6] STEVE GROOM,[6] HARRY TEPLITZ,[6] VANDANA DESAI,[1] DAVID HALE,[9] ROGER SMITH,[9] RICHARD WALTERS,[10] ROBERT QUIMBY,[3] MANSI KASLIWAL,[3] ASSAF HORESH,[3] ERIC BELLM,[3] TOM BARLOW,[3] ADAM WASZCZAK,[11] THOMAS A. PRINCE,[3] AND SHRINIVAS R. KULKARNI[3]

Received 2014 April 04; accepted 2014 May 28; published 2014 July 10

ABSTRACT. The Palomar Transient Factory (PTF) is a multiepochal robotic survey of the northern sky that acquires data for the scientific study of transient and variable astrophysical phenomena. The camera and telescope provide for wide-field imaging in optical bands. In the five years of operation since first light on 2008 December 13, images taken with Mould-R and SDSS-g' camera filters have been routinely acquired on a nightly basis (weather permitting), and two different Hα filters were installed in 2011 May (656 and 663 nm). The PTF image-processing and data-archival program at the Infrared Processing and Analysis Center (IPAC) is tailored to receive and reduce the data, and, from it, generate and preserve astrometrically and photometrically calibrated images, extracted source catalogs, and co-added reference images. Relational databases have been deployed to track these products in operations and the data archive. The fully automated system has benefited by lessons learned from past IPAC projects and comprises advantageous features that are potentially incorporable into other ground-based observatories. Both off-the-shelf and in-house software have been utilized for economy and rapid development. The PTF data archive is curated by the NASA/IPAC Infrared Science Archive (IRSA). A state-of-the-art custom Web interface has been deployed for downloading the raw images, processed images, and source catalogs from IRSA. Access to PTF data products is currently limited to an initial public data release (M81, M44, M42, SDSS Stripe 82, and the *Kepler* Survey Field). It is the intent of the PTF collaboration to release the full PTF data archive when sufficient funding becomes available.

Online material: color figure

1. INTRODUCTION

The Palomar Transient Factory (PTF) is a robotic image-data-acquisition system whose major hardware components include a 92 megapixel digital camera with changeable filters mounted to the 48-inch Palomar Samuel Oschin Telescope. The raison d'être of PTF is to advance our scientific knowledge of transient and variable astrophysical phenomena. The camera and telescope capacitate wide-field imaging in optical bands, making PTF eminently suitable for conducting a multiepochal survey. The Mount Palomar location of the observatory limits the observations to north of $\approx -30°$ in declination. The camera's pixel size on the sky is 1.01″. In the 5 yr of operation since first light on 2008 December 13 (Law et al. 2009), images taken with Mould-R (hereafter R) and SDSS-g' (hereafter g) camera filters have been routinely acquired on a nightly basis (weather permitting), and two different Hα filters were installed in 2011 May (656 and 663 nm). Law et al. (2009) present an overview of PTF initial results and performance, and Law et al. (2010) give an update after the first year of operation. Rau et al. (2009) describe the specific science cases that enabled the preliminary

[1] Spitzer Science Center, California Institute of Technology, Pasadena, CA 91125,; laher@ipac.caltech.edu.
[2] Benoziyo Center for Astrophysics, Weizmann Institute of Science, 76100 Rehovot, Israel.
[3] Division of Physics, Mathematics, and Astronomy, California Institute of Technology, Pasadena, CA 91125.
[4] Department of Physics, University of California, Santa Barbara, CA 93106.
[5] Department of Physics and Astronomy, University of North Carolina, Chapel Hill, NC 27599.
[6] Infrared Processing and Analysis Center, California Institute of Technology, Pasadena, CA 91125.
[7] Department of Physics and Astronomy, The University of Iowa, Iowa City, IA 52242.
[8] Anre Technologies Inc., 3115 Foothill Blvd., Suite M202, La Crescenta, CA 91214.
[9] Caltech Optical Observatories, California Institute of Technology, Pasadena, CA 91125.
[10] Kavli Institute for the Physics and Mathematics of the Universe (WPI), Todai Institutes for Advanced Study, The University of Tokyo, 5-1-5 Kashiwanoha, Kashiwa-shi, Chiba, 277-8583, Japan.
[11] Division of Geological and Planetary Sciences, California Institute of Technology, Pasadena, CA 91125.

planning of PTF observations. The PTF project has been very successful in delivering a large scientific return, as evidenced by the many astronomical discoveries from its data; e.g., Sesar et al. (2012); Arcavi et al. (2010); and van Eyken et al. (2011). As such, it is expected to continue for several more years.

This document presents a comprehensive report on the image-processing and data archival system developed for PTF at the Infrared Processing and Analysis Center (IPAC). A simplified diagram of the data and processing flow is given in Figure 1. The IPAC system is fully automated and designed to receive and reduce PTF data, and generate and preserve astrometrically and photometrically calibrated images, extracted source catalogs and co-added reference images. The system has both software and hardware components. At the top level, it consists of a database and a collection of mostly Perl and some Python and shell scripts that codify the complex tasks required, such as data ingest, image processing and source-catalog generation, product archiving, and metadata delivery to the archive. The PTF data archive is curated by the NASA/IPAC Infrared Science Archive[12] (IRSA). An overview of the system has been given by Grillmair et al. (2010), and the intent of this document is to present a complete description of our system and put forward additional details that heretofore have been generally unavailable.

The software makes use of relational databases that are queryable via structured query language (SQL). The PTF operations database, for brevity, is simply referred to herein as the database. Other databases utilized by the system are called out, as necessary, when explaining their purpose.

Data-structure information useful for working directly with PTF camera-image files, which is important for understanding pipeline processes, is given in § 2. By "pipeline," we mean a scripted set of processes that are performed on the PTF data, in order to generate useful products for calibration or scientific analysis. Significant events that occurred during the project's multiyear timeline are documented in § 3. Our approach to developing the system is given in § 4. The system's hardware architecture is laid out in § 5, and the design of the database schema is outlined in § 6. The PTF-data-ingest subsystem is entirely described in § 7. The tools and methodology we have developed for science data quality analysis (SDQA) are elaborated in § 8. The image-processing pipelines, along with those for calibration, are detailed in § 9. The image-data and source-catalog archive, as well as methods for data distribution to users, are explained in § 10. This paper would be incomplete without reviewing the lessons we have learned throughout the multiyear and overlapping periods of development and operations, and so we cover them in § 11. Our conclusions are given in § 12. Finally, the Appendix presents the simple method of photometric calibration that was implemented prior to when the more sophisticated one of Ofek et al. (2012) was brought into operation.

[12] http://irsa.ipac.caltech.edu/.

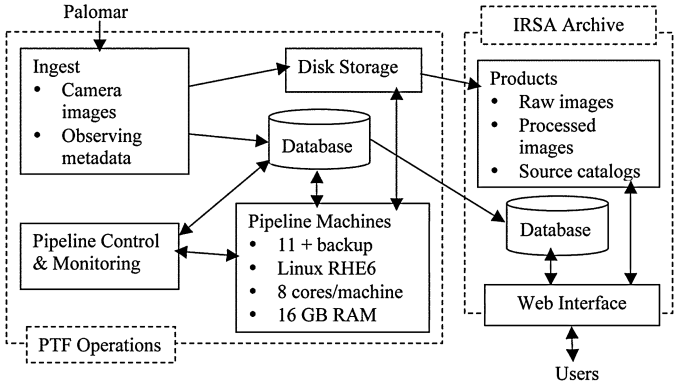

FIG. 1.—Data and processing flow for the IPAC-PTF system.

2. CAMERA-IMAGE FILES

The PTF camera has 12 charge-coupled devices (CCDs) and was purchased from the Canada-France-Hawaii Telescope (Rahmer et al. 2008). The CCDs are numbered $CCDID = 0, \ldots, 11$. Eleven of the CCDs are fully functioning, and one is regrettably inoperable ($CCDID = 3$; there is a broken trace that was deemed too risky to repair). Each CCD has 2048×4096 pixels. The layout of the CCDs in the camera focal plane is 2 rows × 6 columns, where the rows are east-west aligned and the columns north-south. This configuration enables digital imaging of an area approximately $3.45° \times 2.30°$ on the sky (were it not for the inoperable CCD). Law et al. (2009, 2010) give additional details about the camera, system performance, and first results.

PTF camera-image files, which contain the "raw" data, are FITS[13] files with multiple extensions. Each file corresponds to a single camera exposure, and includes a primary HDU (header+data unit) containing summary header information pertinent to the exposure. The primary HDU has no image data, but does include observational metadata, such as where the telescope was pointed, Moon and Sun positional and illumination data, weather conditions, and instrumental and observational parameters. Tables 1 and 2 selectively list the PTF primary-header keywords, many of whose values are also written to the *Exposures* database table during the data-ingest process (see § 6 and § 7). A camera-image file also includes 12 additional HDUs or FITS extensions corresponding to the camera's 12 CCDs, where each FITS extension contains the header information and image data for a particular CCD.

The PTF camera-image data are unsigned 16 bit values that are stored as signed 16 bit integers ($BITPIX = 16$), since FITS does not directly support unsigned integers as a fundamental data type.[14] Thus, the image data values are shifted by 32,768

[13] FITS stands for "Flexible Image Transport System"; see http://fits.gsfc.nasa.gov.

[14] See the CFITSIO User's Reference Guide.

TABLE 1
SELECT KEYWORDS IN THE PTF-CAMERA-IMAGE PRIMARY HEADER

Keyword	Definition
ORIGIN	Origin of data (always "Palomar Transient Factory")
TELESCOP	Name of telescope (always "P48")
INSTRUME	Instrument name (always "PTF/MOSAIC")
OBSLAT	Telescope geodetic latitude in WGS84 (always 33.3574°)
OBSLON	Telescope geodetic longitude in World Geodetic System (WGS) 84 (always −116.8599°)[a]
OBSALT	Telescope geodetic altitude in WGS84 (always 1703.2 m)
EQUINOX	Equinox (always 2000 Julian years)
OBSTYPE	Observation type[b]
IMGTYP	Same as *OBSTYPE*
OBJECT	Astronomical object of interest; currently, always set to "PTF_survey"
OBJRA	Sexagesimal right ascension of requested field in J2000 (HH:MM:SS.SSS)
OBJDEC	Sexagesimal declination of requested field in J2000 (DD:MM:SS.SS)
OBJRAD	Decimal right ascension of requested field in J2000 (degrees)
OBJDECD	Decimal declination of requested field in J2000 (degrees)
PTFFIELD	PTF field number
PTFPID	Project type number
PTFFLAG	Project category flag (either 0 for "non-PTF" or 1 for "PTF" observations)
PIXSCALE	Pixel scale (always 1.01″)
REFERENC	PTF website (always "http://www.astro.caltech.edu/ptf")
PTFPRPI	PTF Project Principal Investigator (always "Kulkarni")
OPERMODE	Mode of operation (either "OCS,"[c] "Manual", or "N/A")
CHECKSUM	Header-plus-data unit checksum
DATE	Date the camera-image file was created (YYYY-MM-DD)
DATE-OBS	UTC date and time of shutter opening (YYYY-MM-DDTHH:MM:SS.SSS)
UTC-OBS	Same as *DATE-OBS*
OBSJD	Julian date corresponding to *DATE-OBS* (days)
HJD	Heliocentric Julian Date corresponding to *DATE-OBS* (days)
OBSMJD	Modified Julian Date corresponding to *DATE-OBS* (days)
OBSLST	Mean local sidereal time corresponding to *DATE-OBS* (HH:MM:SS.S)
EXPTIME	Requested exposure time (s)
AEXPTIME	Actual exposure time (s)
DOMESTAT	Dome shutter status at beginning of exposure (either "open," "closed," or "unknown")
DOMEAZ	Dome azimuth (degrees)
FILTERID	Filter identification number (ID)
FILTER	Filter name (e.g., "*R*", "*g*", "*Ha*656", or "*Ha*663")
FILTERSL	Filter-changer slot position (designated either 1 or 2)
SOFTVER	Palomar software version (Telescope.Camera.Operations.Scheduling)
HDR_VER	Header version

[a] Some FITS headers list this value incorrectly as positive.
[b] Possible setting is "object," "dark," "bias," "dome," "twilight," "focus," "pointing," or "test." Dome and twilight images are potentially useful for constructing flats.
[c] OCS stands for "observatory control system."

data numbers (DN, a.k.a. analog-to-digital units) when read into computer memory ($BZERO = 32768$ is the standard FITS-header keyword that controls the data shifting when the data are read in via a CFITSIO or comparable function), and so the raw-image data are in the 0–65,535 DN range. The raw-image size is 2078×4128 pixels, a larger region than covered by the actual pixels in a CCD because it includes regions of bias overscan "pixels" (which are the data values read out during the pixel sampling time outside of a CCD row or column of detectors).

The *FILTER*, *EXPTIME*, *SEEING*, and *AIRMASS* values associated with camera images are among the variables that have a significant impact on the character and quality of the image data. The exposure time is nominally 60 s, but this is varied as needed for targets of opportunity or reduced to avoid saturation for some targets; e.g., SN 2011fe (Nugent et al. 2011). There is also variation in some of the parameters and imaging properties from one CCD to another (some of the CCDs are better than the others in image-quality terms).

The exposures have GMT time stamps in the camera-image filenames and FITS headers. This conveniently permits all exposures taken in a given night to have the same date of observation (no date boundaries are crossed during an observing night). An example of a typical camera-image filename is

`PTF201108182046_2_o_8242.fits.`

TABLE 2
SELECT KEYWORDS IN THE PTF-CAMERA-IMAGE PRIMARY HEADER (CONTINUED FROM TABLE 1)

Keyword	Definition
SEEING	Seeing full width at half-maximum (FWHM; pixels), an average of FWHM_IMAGE values computed by SExtractor
PEAKDIST	Mean of distance of brightest pixel to centroid pixel (pixels) from SExtractor[a]
ELLIP	Clipped median of ellipticity[b] for all nonextended field objects from SExtractor
ELLIPPA	Mean of ellipse rotation angle (degrees) from SExtractor
FOCUSPOS	Focus position (mm)
AZIMUTH	Telescope azimuth (degrees)
ALTITUDE	Telescope altitude (degrees)
AIRMASS	Telescope air mass
TRACKRA	Telescope tracking speed along R.A. with respect to sidereal time (arcseconds hr^{-1})
TRACKDEC	Telescope tracking speed along decl. with respect to sidereal time (arcseconds hr^{-1})
TELRA	Telescope-pointing right ascension (degrees)
TELDEC	Telescope-pointing declination (degrees)
TELHA	Telescope-pointing hour angle (degrees)
HOURANG	Mean hour angle (HH:MM:SS.SS) based on *OBSLST*
CCD0TEMP	Temperature sensor on $CCDID = 0$ (K)
CCD9TEMP	Temperature sensor on $CCDID = 9$ (K)
CCD5TEMP	Temperature sensor on $CCDID = 5$ (K)
CCD11TEM	Temperature sensor on $CCDID = 11$ (K)
HSTEMP	Heat spreader temperature (K)
DHE0TEMP	Detector head electronics temperature, master (K)
DHE1TEMP	Detector head electronics temperature, slave (K)
DEWWTEMP	Dewar wall temperature (K)
HEADTEMP	Cryogen cooler cold head temperature (K)
RSTEMP	Temperature sensor on radiation shield (K)
DETHEAT	Detector focal plane heater power (%)
WINDSCAL	Wind screen altitude (degrees)
WINDDIR	Azimuth of wind direction (degrees)
WINDSPED	Wind speed (km hr^{-1})
OUTTEMP	Outside temperature (°C)
OUTRELHU	Outside relative humidity fraction
OUTDEWPT	Outside dew point (°C)
MOONRA	Moon right ascension in J2000 (degrees)
MOONDEC	Moon declination in J2000 (degrees)
MOONILLF	Moon illuminated fraction
MOONPHAS	Moon phase angle (degrees)
MOONESB	Moon excess in sky V-band brightness (magnitude)
MOONALT	Moon altitude (degrees)
SUNAZ	Sun azimuth (degrees)
SUNALT	Sun altitude (degrees)

[a]If the value is larger than just a few tenths of a pixel, it may indicate a focus or telescope-tracking problem. There are 33 exposures with failed telescope tracking, acquired mostly in 2009, and their *PEAKDIST* values are generally greater than a pixel.

[b]The ellipticity is from the SExtractor ELLIPTICITY output parameter. The formula A/B in the FITS-header comment should be changed to $1 - B/A$, where A and B are defined in the SExtractor documentation.

Embedded in the filename is the date concatenated with four digits of the fractional day. The next filename field is the filter number. The next field is a one-character moniker for the image type: "o" stands for "object," "b" stands for "bias," "k" stands for "dark," etc. The last field before the ".fits" filename extension is a nonunique counter, which is reset to zero when the camera is rebooted (which can happen in the course of a night, although infrequently).

3. SIGNIFICANT PROJECT EVENTS

There were three different events that occurred during the course of the project that affected how the processing is done and how the results are interpreted. There was a fourth event, which occurred last, that is mostly programmatic in nature. It is convenient to view these events as having transpired during the day, in between nightly data-taking periods.

On 2009 October 9, the camera electronics were reconfigured, which greatly improved the camera's dynamic range, thus raising the DN levels at which the pixel detectors saturate. Image data taken up to this date saturate in the 17,000–36,000 DN range, depending on the CCD. After the upgrade, the data saturation occurs in the 49,000–55,000 DN range. Table 3 lists the CCD-dependent saturation values, before and after the upgrade.

TABLE 3
CCD-dependent Saturation Values, before and after the PTF-Camera-Electronics Upgrade, Which Occurred on 2009 October 9

CCDID	Before (DN)	After (DN)
0	34,000	53,000
1	36,000	54,000
2	25,000	55,000
3	N/A	N/A
4	31,000	49,000
5	33,000	50,000
6	26,000	55,000
7	17,000	55,000
8	42,000	53,000
9	19,000	52,000
10	25,000	52,000
11	36,000	53,000

On 2010 July 15, the positions of the R and g filters were swapped in the filter wheel. This not only made the expected filter positions in the filter wheel time dependent, but also altered the positions of the ghost reflections on the focal plane (and, hence, in the images).

On 2010 September 2, the "fogging problem" was solved, which had been causing a diffuseness in the images around bright stars, and was the result of an oil film slowly building up on the camera's cold CCD window during the times between the more-or-less bimonthly window cleanings. Ofek et al. (2012) discuss the resolution of this problem in more detail.

On 2013 January 1, the official PTF program ended and the "intermediate" PTF (iPTF) program started.[15] Coincidently, PTF-archive users will notice that DAOPHOT source catalogs (Stetson 1987) are available from this point on, in addition to the already available SExtractor source catalogs (Bertin 2006a), which is the result of pipeline upgrades that were delivered around that time. Also, this was around the time that the IPAC-PTF reference-image, real-time, and difference-image pipelines came online.

4. DEVELOPMENT APPROACH

This section covers our design philosophy and assumptions and the software guidelines that we followed in our development approach.

4.1. Design Philosophy and Assumptions

The development of the data-ingest, image-processing, archival, and distribution components for PTF data and products have leveraged existing astronomical software and the relevant infrastructure of ongoing projects at IPAC.

Database design procedures developed at IPAC have been followed in order to keep the system as generic as possible and not reliant on a particular brand of database. This allows the flexibility of switching from one database to another over the project's many years of operation, as necessary.

We strived for short database table and column names to minimize keyboard typing (and mostly achieved this) and to quicken learning the database schema. We avoided renaming primary keys when used as foreign keys in other tables, in order to keep table joins simple. (A primary key is a column in a table that stores a unique identification number for each record in the table, and a foreign key is a column in a table that stores the primary key of another table and serves to associate a record in one table with a record in another table.)

The metadata stored in the database on a regular basis during normal operations come directly from, or are derivable from, information in either the header or filename of camera-image files containing the raw data, as well as nightly-observing metadata files. Thus, very little prior information about scheduling of specific observations is required.

We expect to have to be able to deal with occasional corrupt or incomplete data. The software must therefore be very robust, and, for example, be able to supply missing information, if possible. Having the ability to flag bad data in various ways is useful. This and the means of preventing certain data from undergoing processing are necessary parts of the software and database design.

Another important aspect of our design is versioning. Software, product, and archive versioning are handled independently in our design, and this simplifies the data and processing management. A data set, for example, may be subjected to several rounds of reprocessing to smooth out processing wrinkles before its products are ready to be archived.

4.2. Software Guidelines

An effort has been made to follow best programming practices. A very small set of guidelines were followed for the software development, and no computer-language restrictions were imposed so long as the software met performance expectations. We have made use of a variety of programming languages in this project, as our team is quite diverse in preferences and expertise.

The source code is checked into a version control system (CVS). An updated CVS version string is automatically embedded into every source-code file each time a new file version is checked into the CVS repository, and this facilitates tracking deployed software versions when debugging code. The Web-based software-version-control system called *GNATS* is used for tracking software changes and coordinating software releases.

All Perl scripts are executed from a common installation of Perl that is specified via environment variable *PERL_PATH* and require explicit variable declaration ("use strict;"). Minimal use

[15] http://ptf.caltech.edu/iptf/.

is made of global variables. Stand-alone blocks of code are wrapped as subroutines and put into a library for reuse and complexity hiding.

Modules requiring fast computing speed were generally developed in the C language on Mac laptops and tested there prior to deployment on the Linux pipeline machines. Thus, the software benefited from multiplatform testing, which enhances its robustness and improves the chances of uncovering bugs.

All in-house software, which excludes third-party software, is designed to return a system value in the 0–31 range for normal termination, in the 32–63 range for execution with warnings, and $> = 64$ if an error occurs. At the discretion of the programmer, specific values are designated for special conditions, warnings, and errors that are particular to the software under development.

All scripts generate log files that are written to the PTF logs directory, which is appropriately organized into subdirectories categorized by process type. The log files are very verbose, and explicit information is given about the processes executed, along with the input parameters and command-line options and switches used. Software version numbers are included, as well as is timing information, which is useful for benchmark profiling.

5. SYSTEM ARCHITECTURE

Figure 2 shows the principal hardware components of the IPAC-PTF system, which are located on the Caltech campus. Firewalls, servers, and pipeline machines, which are depicted as rectangular boxes in the figure, are currently connected to a 10 gigabit s^{-1} network (this was upgraded in 2012 from 1 gigabit s^{-1}). Firewalls provide the necessary security and isolation between the PTF transfer machine that receives nightly PTF data, the IRSA Web services, and the operations and archive networks. A demilitarized zone (DMZ) outside of the inner firewall has been set up for the PTF transfer machine. A separate DMZ exists for the IRSA search engine and Web server.

The hardware has redundancy to minimize downtime. Two data-ingest machines, a primary and a backup, are available for the data-ingest process (see § 7), but only one of these machines is required at any given time. There are 12 identical pipeline machines for parallel processing, but only 11 are needed for the pipelines, and so the remaining machine serves as a backup. The pipeline machines have 64 bit Linux operating systems installed (Red Hat Enterprise 6, upgraded from 5 in early 2013), and each has eight CPU cores and 16 Gbyte (GB) of memory. There are two database servers: a primary for regular PTF operations and a secondary for the database backup. Currently, the database servers are running the Solaris-10 operating system, but are accessible by database clients running under Linux.

There is ample disk space, which is attached to the operations file server, for staging camera-image files during the data ingest and temporarily storing pipeline intermediate and final

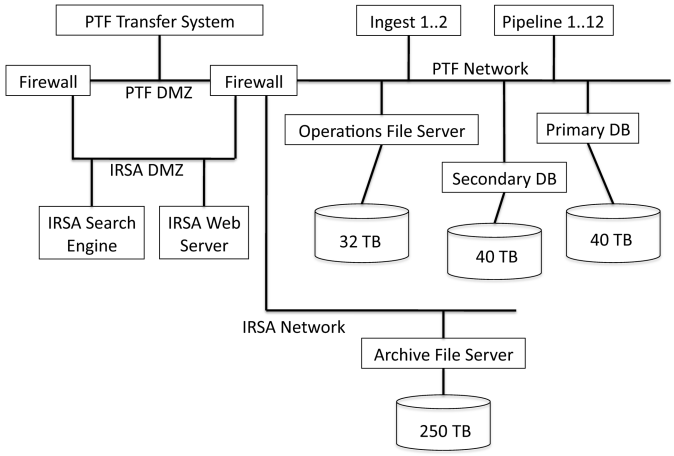

FIG. 2.—Computing, network, and archiving hardware for the IPAC-PTF system.

products. These disks, which are called sandboxes, are cross-mounted to all pipeline machines for the pipeline image processing. This design strategy minimizes network traffic by allowing intermediate products to be available for a short time for debugging purposes and only transferring final products to the archive. The IRSA archive file server is set up to allow the copying of files from PTF operations through the firewall. The IRSA archive disk storage is currently 250 Tbyte (TB), and this will be augmented as needed over the project lifetime. It is expected that this disk capacity will be doubled by the end of the project. In general, the multi-terabyte disk storage is broken up into 8 TB or 16 TB partitions to facilitate disk management and file backups.

6. DATABASE

We initially implemented the database in Informix to take advantage of Informix tools, interfaces, methodologies, and expertise developed under the Spitzer project. After a few months, we made the decision to switch to an open-source PostgreSQL database, as our Informix licensing did not allow us to install the database server on another machine and purchasing an additional license was not an option due to limited funding. All in all, it was a smooth transition, and there was a several-month period of overlap where we were able to switch between Informix and PostgreSQL databases simply by changing a few environment variables.

Figure 3 depicts the database schema for the basic tables associated with ingesting PTF data. Some of the details in the figure are explained in its caption and in § 7. Briefly, the *Nights* database table tracks whether any given night has been successfully ingested ($status = 1$) or not ($status = 0$). A record for each camera exposure is stored in the *Exposures* database table, and each record includes the camera-image filename, whether the exposure is good ($status = 1$) or not ($status = 0$, such as in the rare case of bad sidereal tracking), and other

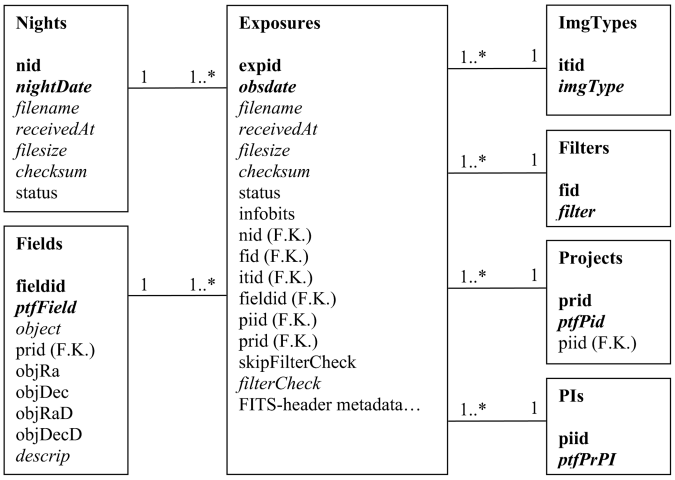

FIG. 3.—IPAC-PTF database-schema design for the data ingest (see § 7). The database table name is given at the top of each box. The bold-font database column listed after the table name in each box is the primary key of the table. The columns listed in bold-italicized font are the alternate keys. The columns listed in regular font are not-null columns, and in regular-italicized font are null columns (which are columns in which null values possibly may be stored). "F.K." stands for foreign key, and "1 1..*" stands for one record to many records, etc.

exposure and data-file metadata. The exposure metadata is obtained directly from the primary FITS header of the camera-image file (see § 2). The remaining database tables in the figure track the database-normalized attributes of the exposures. The *Filters* database table, for example, contains one record per unique camera filter used to acquire the exposures.

Not shown in Figure 3 is the *FieldCoverage* database table, which contains the most complete set available of fields to be scheduled for multiepochal observation, whereas all other tables for information about PTF data store only records for data that have already been acquired. This table is not required for the data ingest, but is used by the pipeline that performs the astrometric calibration (see § 9.15), since it includes columns that identify cached astrometric catalogs for each PTF field. A fairly complete list of PTF-operations database tables is given in Table 4.

Figure 4 shows a portion of the database schema relevant to the pipeline image processing. The key features of the database tables involved are given in the remainder of this section. The various utilities of these database tables are discussed throughout this paper as well. For conciseness, several equally important database tables are not shown, but are discussed presently (e.g., see § 9.15). These include tables for science data quality analysis (SDQA), photometric calibration, and tracking artifacts such as ghosts and halos.

The *Pipelines* database table assigns a unique index to each pipeline and stores useful pipeline metadata, such as their priority order of execution. See § 9.1 and § 9.5 for a detailed discussion of the table's data contents.

The *RawImages* database table stores metadata about raw images, one record per raw-image file, where each raw-image file corresponds to the data from one of the camera's CCDs in an exposure. While the 12 CCD camera images are archived (and tracked in the *Exposures* database table), the raw-image files associated with the *filename* column in the *RawImages* database table are not archived, but serve as pipeline inputs from the sandbox, for as long as they are needed, and then are eventually removed from the file system to avoid duplicate storage.

The *ProcImages* database table stores metadata about processed images, one record per image file. There is a one-to-many relationship between *RawImages* and *ProcImages* records because a given raw image can be processed multiple times, which is useful when the software version (tracked in the *SwVersions* database table) is upgraded or the software configuration (tracked in the *CdfVersions* database table) needs to be changed. Moreover, a given raw image can be processed by different pipelines. The *version* column keeps track of the processing episode for a given combination of raw image (*rid*) and pipeline (*ppid*). The *vBest* column is automatically set to one for the latest version and zero for all previous versions, unless a previous version has the column set to $vBest = 2$, in which case it is "locked" on that previous version. In addition, similar products can be generated by different pipelines, and the *pBest* column flags which of the pipelines' products are to be archived.

The *Catalogs* database table stores metadata about the extracted source catalogs, one record per catalog file. There is a one-to-many relationship between *ProcImages* and *Catalogs* records because catalogs can be regenerated from a given processed image multiple times. Image processing takes much more time than catalog generation, and the latter can be redone, if necessary, without having to redo the former. The structure of the *Catalogs* database table is analogous to that of the *ProcImages* database table with regard to product versioning and tracking.

The *AncilFiles* database table stores metadata about ancillary files that are created during the pipeline image processing and directly related to processed images (i.e., ancillary files besides catalogs, which are a special kind of ancillary file registered in the *Catalogs* database table). Ancillary files presently include data masks and JPEG preview images, which are distinguished by the *ancilType* column. The table is flexible in that new *ancilType* settings can be defined for new classes of ancillary files that may arise in the course of development. This database table enforces the association between all ancillary files and their respective processed images.

Calibration files are created by calibration pipelines and applied by image-processing pipelines. The *CalFiles* and *CalFileUsage* database tables allow multiple versions of calibration files to be tracked and associated with the resulting processed images.

The *ArchiveVersions* database table is pivotal for managing products in the data archive. For more on that and the archive-related columns in the *ProcImages*, *Catalogs*, *AncilFiles*, *CalFiles*, and *CalAncilFiles* database tables, see § 10.1.

TABLE 4

OPERATIONS DATABASE TABLES OF THE PALOMAR TRANSIENT FACTORY

Table name	Description
Nights	Nightly data-ingest status and other metadata (e.g., images-manifest filenames). Unique index: *nid*. Alternate key: *nightdate*.
Exposures	Exposure status and other metadata (e.g., camera-image filenames). Unique index: *expid*. Alternate key: *obsdate*.
CCDs	CCD constants (e.g., sizes of raw and processed images, in pixels). Unique index: *ccdid*.
Fields	Observed PTF field positions and their assigned identification numbers (IDs). Unique index: *fieldid*. Alternate key: *ptffield*.
FieldCoverage	Field positions and their fractional overlap onto SDSS[a] fields. Unique index: *fcid*. Alternate keys: *ptffield* and *ccdid*.
ImgTypes	Image types taken by PTF camera ("object," "bias," "dark," etc.). Unique index: *itid*.
Filters	Camera filters available. Currently R, g, and two different Hα filters are available. Unique index: *fid*.
FilterChecks	Cross-reference table between filter-checker output indices and human-readable filter-check outcomes.
PIs	Principal-investigator contact information. Unique index: *piid*.
Projects	Project abstracts, keywords, and associated investigators. Unique index: *prid*.
Pipelines	Pipeline definitions and pipeline-executive metadata (e.g., *priority*). Unique index: *ppid*.
RawImages	Raw-image metadata (after splitting up FITS-multiextension camera images as needed). Unique index: *rid*.
ProcImages	Processed-image metadata (e.g., image filenames). Unique index: *pid*. Alternate keys: *rid*, *ppid*, and *version*.
Catalogs	Metadata about SExtractor and DAOPHOT catalogs extracted from processed images. Unique index: *catid*.
AncilFiles	Ancillary-product associations with processed images. Unique index: *aid*. Alternate keys: *pid* and *anciltype*.
CalFiles	Calibration-product metadata (e.g., filenames, and date ranges of applicability). Unique index: *cid*.
CalFileUsage	Associations between processed images (*pid*) and calibration products (*cid*).
CalAncilFiles	Ancillary-calibration-product metadata. Unique index: *caid*. Alternate keys: *cid* and *anciltype*.
IrsaMeta	Processed-image metadata required by IRSA (e.g., image-corner positions). Unique index: *pid* (foreign key).
QA	Quality-analysis information (e.g., image statistics). Unique index: *pid* (foreign key).
AbsPhotCal	Absolute-photometric-calibration coefficients. Unique index: *apcid*. Alternate keys: *nid*, *ccdid*, and *fid*.
AbsPhotCalZpvm	Zero-point-variability-map data. Primary keys: *apcid*, *indexi*, and *indexj*.
RelPhotCal	Relative-photometric-calibration zero points. Unique index: *rpcid*. Alternate keys: *ptffield*, *ccdid*, *fid*, and *version*.
RelPhotCalFileLocks	Utilizes row locking to manage file locking. Primary keys: *ptffield*, *ccdid*, and *fid*.
Ghosts	Metadata about ghosts in processed images. Unique index: *gid*. Alternate keys: *pid*, *ccdid*, *fid*, and (x, y).
Halos	Metadata about halos in processed images. Unique index: *hid*. Alternate keys: *pid*, *ccdid*, *fid*, and (x, y).
Tracks	Metadata about aircraft/satellite tracks in processed images. Unique index: *tid*. Alternate keys: *pid*, *ccdid*, *fid*, and *num*.
PSFs	Point spread functions (PSFs) in DAOPHOT format. Unique index: *psfid*. Alternate key: *pid*.
RefImages	Reference-image metadata (e.g., filenames). Unique index: *rfid*. Alternate keys: *ccdid*, *fid*, *ptffield*, *ppid*, and *version*.
RefImageImages	Associations between processed images (*pid*, $ppid = 5$) and reference images (*rfid*, $ppid = 12$).
RefImAncilFiles	Ancillary-product associations with reference images. Unique index: *rfaid*.
RefImageCatalogs	Metadata about SExtractor and DAOPHOT catalogs extracted from reference images. Unique index: *rfcatid*.
IrsaRefImMeta	Reference-image metadata required by IRSA (e.g., image-corner positions). Unique index: *rfid* (foreign key).
IrsaRefImImagesMeta	IRSA-required metadata for processed images that are co-added to make the reference images (see *RefImageImages* database table).
SDQA_Metrics[b]	SDQA-metric definitions. Unique index: *sdqa_metricid*.
SDQA_Thresholds	SDQA-threshold settings. Unique index: *sdqa_thresholdid*.
SDQA_Statuses	SDQA-status definitions. Unique index: *sdqa_statusid*.
SDQA_Ratings	SDQA-rating values for processed images. Unique index: *sdqa_ratingid*. Alternate keys: *pid* and *sdqa_metricid*.
SDQA_RefImRatings	SDQA-rating values for reference images. Unique index: *sdqa_refimratingid*. Alternate keys: *rfid* and *sdqa_metricid*.
SDQA_CalFileRatings	SDQA-rating values for calibration files. Unique index: *sdqa_calfileratingid*. Alternate keys: *cid* and *sdqa_metricid*.
SwVersions	Software version information. Unique index: *svid*.
CdfVersions	Configuration-data-file version information. Unique index: *cvid*.
ArchiveVersions	Metadata about archive versions. Unique index: *avid*.
DeliveryTypes	Archive delivery-type definitions. Unique index: *dtid*.
Deliveries	Archive delivery-tracking information. Unique index: *did*.
Jobs	Pipeline-job tracking information. Unique index: *jid*.
ArchiveJobs	Archive-job tracking information. Unique index: *ajid*.
JobArbitration	Job-lock table.
IRSA	Temporary table for marshaling of metadata to be delivered to the IRSA archive.

[a] Sloan Digital Sky Survey (York et al. 2000)

[b] SDQA stands for science data quality analysis.

The *Jobs* database table is indexed by primary key *jid*. It contains a number of foreign keys that index the associated pipeline (*ppid*) and various data parameters (e.g., night, CCD, and filter of interest). It contains time-stamp columns for when the pipeline started and ended, as well as elapsed time, and it also contains columns for pipeline exit code, status, and machine number. Possible status values -1, 0, or 1 indicate the job is suspended, is ready to be executed, or has been executed, respectively.

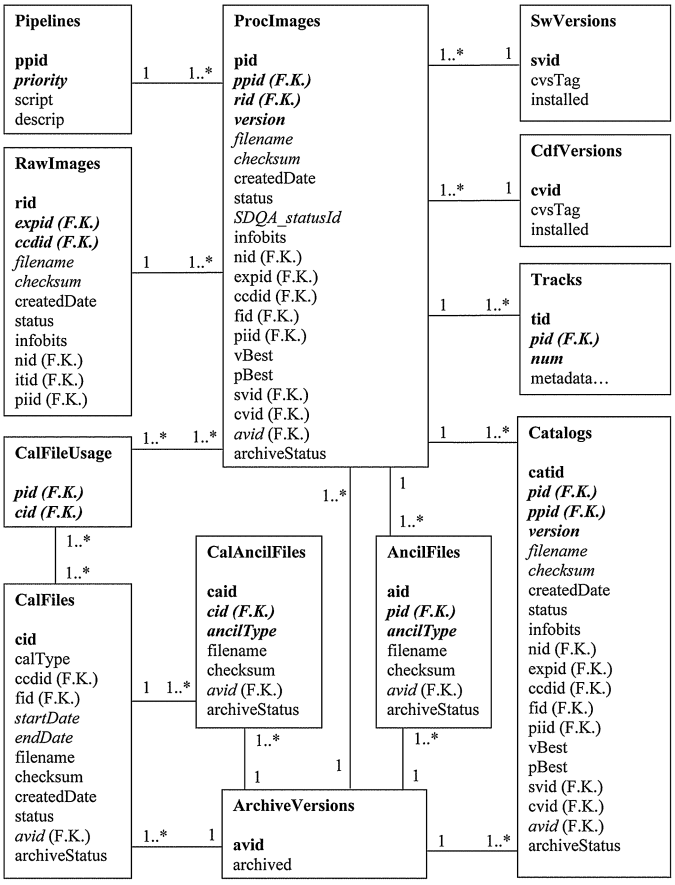

FIG. 4.—IPAC-PTF database-schema design for the pipeline image processing (see § 9). The figure nomenclature is explained in the caption of Fig. 3.

The *ArchiveJobs* database table is indexed by primary key *ajid*. Since product archiving is done on a nightly basis, the database table has columns that store the date of the night of interest (*nightDate*), and the associated night database index (foreign key *nid*) for added convenience. It contains time-stamp columns for when the archive job started and ended, as well as for the elapsed time, and it also contains columns for the archive-job status and virtual-pipeline-operator exit code (see § 9.6). Possible status values -1, 0, or 1 indicate the job is either in a long transaction (currently running or temporarily suspended), is ready to be executed, or has been executed, respectively.

All database tables that store information about files have a column for storing the file's checksum; this is useful for verifying the data integrity of the file over time. There is also the very useful *status* column for tracking whether the file is good ($status = 1$) or not ($status = 0$); many pipeline database queries for files require $status > 0$, and files with $status = 0$ are effectively removed from the processing. Note also that the filename column in these tables is for storing the full path and filename, in order to completely specify the file's location in file storage. Most of the database tables in the schema have their first column data-typed as a database serial identification number, in order to enforce record-index uniqueness, and this is called the primary key of the database table.

The database is backed up weekly, and generally at a convenient time, i.e., when the pipelines are not running. The procedure involves stopping all processes that have database connections (e.g., the pipeline-executive jobbers) because it is desirable to ensure the database is in a known state when it is backed up. A script is run to query for database-validation data. The database server is stopped, and the database file system is snapshotted. This step takes just a few seconds, and the database server and pipelines can be restarted immediately afterwards. This backup procedure is performed by the pipeline operator. The database administrator is then responsible for building a copy of the database from the snapshot and validating it. The database copy is made available to expert users from a different database server. It is sometimes expedient to test software for schema and data content changes in the users' database prior to deployment in operations.

7. DATA INGEST

This section describes the data flow, processes, and software involved in the nightly ingestion of PTF data at IPAC. The data-ingest software has been specially developed in house for the PTF project.

A major requirement is that the ingest process shall not modify either the camera-image filenames as received or the data contained within the files. The reason for this is to ensure traceability of the files back to the mountain top where they are created. Moreover, there are opportunities to ameliorate the image metadata in the early pipeline processing, if needed, and experience has shown that, in fact, this must be done occasionally. The ingest principal functions are to move the files into archival disk storage and store information about them in a relational database. There are other details, and these are described in the subsections that follow.

7.1. High-Level Ingest Process

PTF camera-image files are first sent to a data center in San Diego, CA from Mount Palomar via fast microwave link and landline as an intermediate step, and then pushed to IPAC over the Internet. The files are received throughout the night at IPAC onto a dedicated data-transfer computer that sits in the IPAC DMZ (see § 5). A mirrored 1 TB disk holds the */inbox* partition where the files are written upon receipt. This partition is exported via network file system (NFS) to both primary and backup data-ingest machines, which are located behind the firewall. The primary machine predominantly runs the data-ingest processes. There is also a separate backup data-ingest computer in case the primary machine malfunctions, and this machine is

also utilized as a convenience for sporadically required manual data ingestion.

A file containing a cumulative list of nightly image files, along with their file sizes and Message-digest algorithm 5 (MD5) checksums, is also updated throughout the night and pushed to IPAC after every update. This special type of file, each one uniquely named for the corresponding night, is called the "images manifest." The images manifest has a well-defined filename with embedded observation date and fixed filename extension, suitable for parsing via computer script. An end-of-file marker is written to the images manifest at the end of the night after all image files have been acquired and transferred. This signals the IPAC data-ingest software subsystem that an entire night's worth of data has been received, and the data-ingest process is ready to be initiated for the night at hand. The contents of each images manifest are essentially frozen after the end-of-night marker has been written.

The basic data-ingest process involves copying all image files to archival spinning disk and registering metadata about the night and image files received in the database. A number of steps are involved, and these steps foremost include verifying that the image files are complete, uncorrupted, permanently stored, and retrievable for image processing.

The data are received into disk subdirectories of the */inbox* partition, each named for the year, month, and day of the observations. The date and time stamps in the data are in GMT. A cron job running on the data-ingest computer every 30 minutes launches a Bourne shell script, called *automate_stage_ingest*, that checks for both the existence of the images manifest of the current night and that the end-of-night signal is contained in the images manifest. A unique lock file is written to the /tmp directory to ensure that only one night at a time is ingested. It then initiates the high-level data-ingest process after these conditions are met. This process runs under the root account because file ownership must be changed from the data-transfer account to the operations account under which the image-processing pipelines are executed.

The high-level data-ingest process is another Bourne shell script, called *stage_PTF_raw_files*, that performs the following steps:

1. Checks that the number of files received matches the number of files listed in the images manifest. An alert is e-mailed to operations personnel if this condition is not satisfied, and the process is halted. The cron job will try again 30 minutes later for the current night.

2. Copies the files into an observation-date-stamped subdirectory under the */staging* partition, which is owned by the operations account and is an NFS mount point from the operations file server.

3. Changes to the aforementioned data directory that houses the nightly files to be ingested, and executes the low-level data-ingest processes (see § 7.2). Bourne-shell script *ingest_staged_fits_files* wraps the commands for these processes.

4. As a file backup, copies the files into an observation-date-stamped subdirectory under the */nights* partition, which is also owned by the operations account, but is an NFS mount point from the archive file server. This is done in parallel to the low-level data-ingest process, so as not to hold it up.

5. Checks the MD5 checksums of the files stored in the observation-date-stamped subdirectory under the */nights* partition. Again, this rather time-consuming process is done in parallel to the low-level data-ingest processes.

6. Removes the corresponding subdirectory under the */inbox* partition (and all files therein) upon successful data ingest. This will inhibit the cron job from trying to ingest the same night again.

As a final step, the aforementioned script *ingest_staged_fits_files* executes a database command that preloads camera-image-splitting pipelines for the current night into the *Jobs* database table, one pipeline instance per camera-image file. This pipeline is described in § 9.10.

All scripts generate log files that are written to the *scripts* and *ingest* subdirectories in the PTF logs directory.

7.2. Low-Level Ingest Processes

There are three low-level data-ingest processes, which are executed in the following order:

1. Ingest the camera-image files;
2. Check the file checksums; and
3. Ingest the images manifest.

These processes are described in detail in the following paragraphs.

The Perl script called *ingestCameraImages.pl* works sequentially on all files in the current working directory (an observation-date-stamped subdirectory under the */staging* partition). A given file first undergoes a number of checks. Files that are not FITS files or less that 5 minutes old are skipped for the time being. All files that are FITS files and older than 5 minutes are assumed to be PTF camera-image files and will be ingested. The MD5 checksum is computed, and the file size is checked. Files smaller than 205 Mbyte will be ingested, but the *status* column will be set to zero and bit $2^0 = 1$ will be set in the *infobits* column of the *Exposures* database table (see Table 5) for records associated with files that are smaller than expected, as this has revealed an upstream software problem in the past. Select keywords are read from the FITS header (i.e., a large subset of the keywords listed in Tables 1 and 2). The temperature-related FITS keywords are expected to be missing immediately after a camera reboot, in which case the software substitutes the value zero for these keywords, and bit $2^9 = 512$ is set in the *infobits* column of the *Exposures* database table. Files with missing *FILTER*, *FILTERID*, or *FILTERSL* will have both their values and their *status* set to zero in the *Exposures* database table, along with bit $2^2 = 4$ set in the *infobits* column

TABLE 5

BITS ALLOCATED FOR FLAGGING DATA-INGEST CONDITIONS AND EXCEPTIONS IN THE *INFOBITS* COLUMN OF THE *EXPOSURES* DATABASE TABLE

Bit	Definition
0	File size too small
1	*IMGTYP* = "object" and *DOMESTAT* = "closed"
2	$FILTER = 0$, $FILTERID = 0$ and/or $FILTERSL = 0$
4	Sidereal-tracking failure (manually set after image inspection)
6	Checksum mismatch: database vs. images manifest
7	Checksum mismatch: recomputed vs. images manifest
8	File-size mismatch: recomputed vs. images manifest
9	One or more noncrucial keywords missing

of the *Exposures* database table. All science-image files are checked for the unlikely state of an unopened telescope dome (i.e., *IMGTYP* = "object" and *DOMESTAT* = "closed"), in which case the associated *status* column is set to zero and bit $2^1 = 2$ is set in the *infobits* column of the *Exposures* database table. The file is then copied from the */staging* partition to the appropriate branch of the observation-date-based directory tree in the camera-image-file archive. A record is inserted into the *Exposures* database table for the ingested file, and, if necessary and usually at a lower frequency, new records are inserted into the following database tables: *PIs*, *Projects*, *Nights*, *Filters*, *ImgTypes*, and *Fields*. For example, Table 6 lists the possible PTF-image types that are ingested and registered in the *ImgTypes* database table. Finally, the ingested file is removed from the current working directory, and the software moves on to ingest the next file. The process terminates after all FITS files have been ingested.

The Perl script called *checkIngestedCameraImages.pl* recomputes the MD5 checksums of archived PTF camera-image files, and, for each file, compares the checksum with that stored in the database and in the images manifest. This script can be run any time there is a want or need to check data-file integrity for a given night. The associated *Exposures* database record is updated with $STATUS = 0$ in the rare event of checksum mismatch, and the appropriate bit in the *infobits* column is set (see Table 5).

The Perl script called *ingestImagesManifest.pl* copies the images manifest to its appropriate archival-disk nightly subdirectory and registers its location and filename in the *Nights* database table, along with relevant metadata, such as MD5 checksum, file size, status, and database-record-creation time stamp.

TABLE 6

PTF-IMAGE TYPES IN THE *IMGTYPES* DATABASE TABLE

itid	IMGTYP
1	object
2	dark
3	bias
4	dome
5	twilight
6	focus
7	pointing
8	test

8. SCIENCE DATA QUALITY ANALYSIS

SDQA is an integral part of the design implemented for PTF, which is outlined by Laher et al. (2009) in the context of a different ground-based project under proposal. It is necessary to provide some details about the IPAC-PTF SDQA subsystem at this point, so that interactions between it and the pipelines can be more fully understood.

Typically within hours after a night's worth of camera images have been ingested and the camera-image-splitting pipelines have been executed (see § 9.10), the camera images are inspected visually for problems. The preview images generated by the camera-image-splitting pipelines play a pivotal part in speeding up this task. An in-house Web-based graphical user interface (GUI) has been designed and implemented to provide basic SDQA functionality (see Fig. 5), such as displaying previews of raw and processed images, and dynamically generating time-series graphs of SDQA quantities of interest. The source code for the GUI and visual-display software tools have been developed in Java, primarily for its platform-independent and multithreading capabilities. The software queries the database for its information. The Google Web Toolkit[16] has been used to compile the Java code into Javascript for relatively trouble-free execution under popular Web browsers. The GUI has drill-down capability to selectively obtain additional information. The screen shot in Figure 5 shows the window that displays previews of PTF camera images and associated metadata. The previews load quickly and have sufficient detail to inspect the nightly observations for problems and assess the data quality (e.g., when investigating astrometric-calibration failures). In the event of telescope sidereal-tracking problems, which are spotted visually in the GUI (and occur infrequently), the associated *status* column is set to zero and bit $2^4 = 16$ is set in the *infobits* column of the *Exposures* database table (see Table 5).

A major function of our SDQA subsystem is to compute and store in the database all the needed quantities for assessing data quality. The goal is to boil down questions about the data into relatively simple or canned database queries that span the parameter space of the data on different scales. Having a suitable framework for this in place makes it possible to issue a variety of manually requested and automatically generated reports. During pipeline image processing, SDQA data are computed for the images and astronomical sources extracted from the images and utilized to grade the images and sources. The reports summarize the science data quality in various ways and provide feedback to telescope, camera, facility, observation-scheduling, and data-processing personnel.

[16] http://www.gwtproject.org.

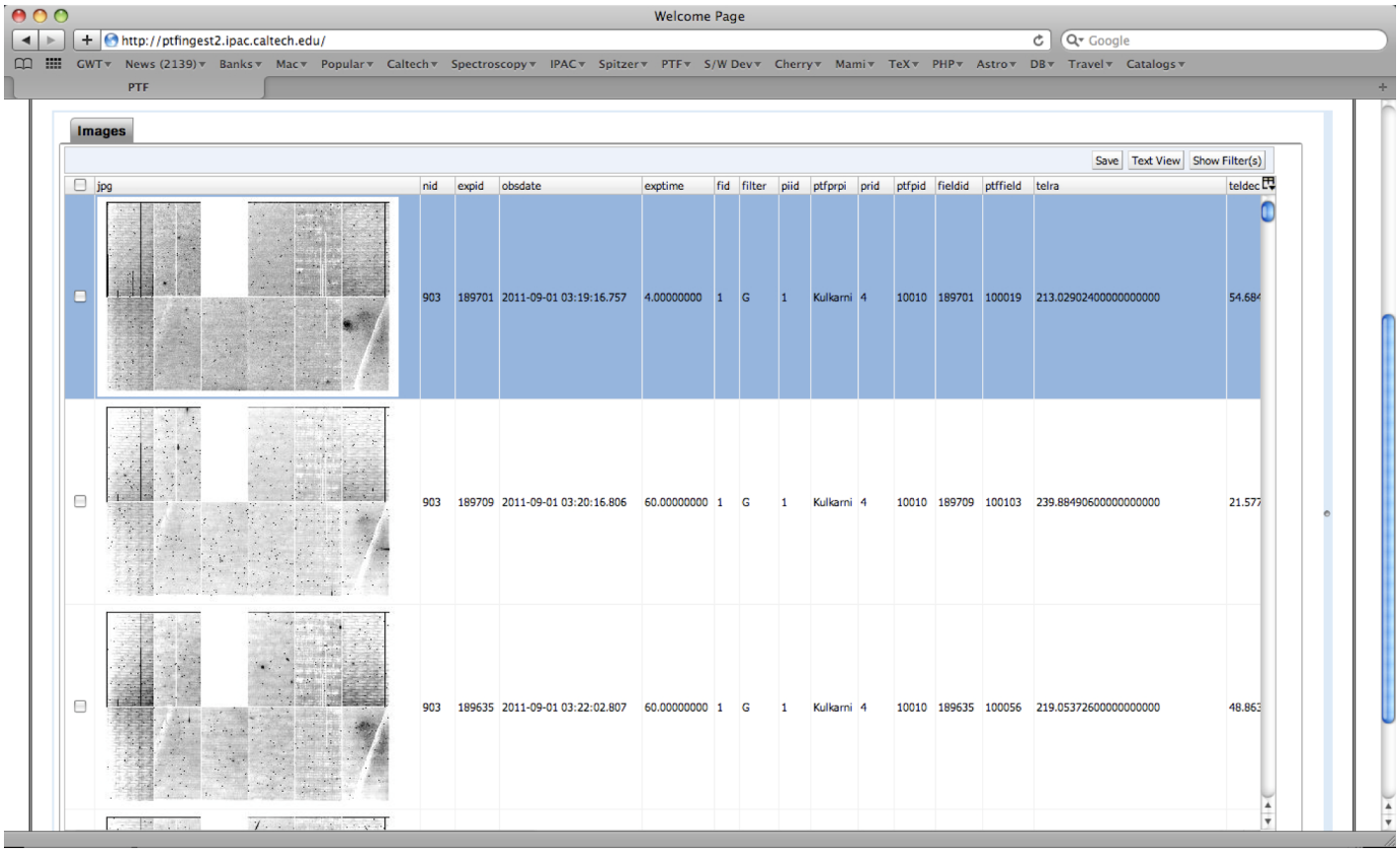

FIG. 5.—Sample screen shot of the SDQA GUI developed for the IPAC-PTF system. See the electronic edition of the *PASP* for a color version of this figure.

Figure 6 shows our SDQA database-schema design for processed images. Note that the design is easily extended for other pipeline products. The *ProcImages* database table is indexed by *pid* and stores metadata about processed images, including the *sdqa_statusid*, which is an integer that indexes the SDQA grade assigned to an image. A processed image is associated with both a raw image (*rid*) and a pipeline (*ppid*). As the pipeline software is upgraded, new versions of a processed image for a given raw image and pipeline will be generated, and, hence, a *version* column is included in the table to keep track of the versions. The *vBest* column flags which version is best; there is only one best version and it is usually the latest version.

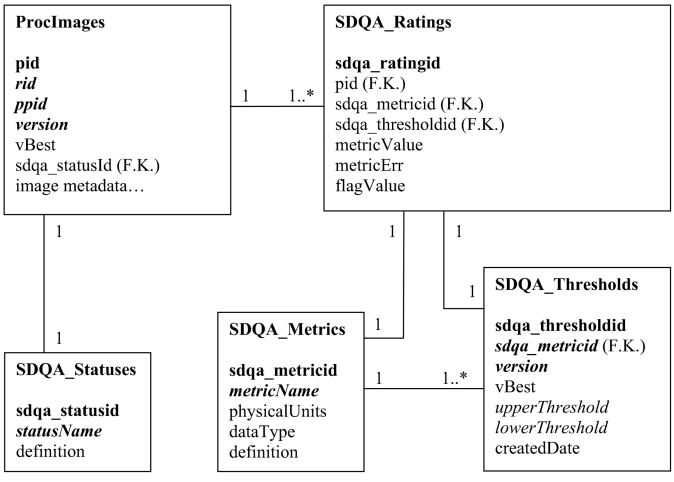

FIG. 6.—IPAC-PTF SDQA database-schema design. The figure nomenclature is explained in the caption of Fig. 3.

SDQA metrics are diverse, predefined measures that characterize image quality; e.g., image statistics, astrometric and photometric figures of merit and associated errors, and counts of various things, like extracted sources. The *SDQA_Metrics* database table stores the SDQA metrics defined for IPAC-PTF operations, and these are listed in Tables 7 through 8. The imageZeroPoint SDQA metric (metricId $= 48$) is set to NaN (not a number) in the database if either (1) the image did not overlap an SDSS field; (2) there were an insufficient number of Sloan Digital Sky Survey (SDSS) sources; or (3) the filter used for the exposure was neither g nor R band (only these two PTF bands are photometrically calibrated at this time).

SDQA thresholds can be defined for values associated with SDQA metrics. The *SDQA_Thresholds* database table stores the SDQA thresholds defined for IPAC-PTF operations and

TABLE 7

IPAC-PTF SDQA METRICS STORED IN THE *SDQA_METRICS* DATABASE TABLE

metricId	metricName	physicalUnits	Definition
1	nGoodPix	Counts	Number of good pixels.
2	nDeadPix	Counts	Number of dead pixels.
3	nHotPix	Counts	Number of hot pixels.
4	nSpurPix	Counts	Number of spurious pixels.
5	nSatPix	Counts	Number of saturated pixels.
6	nObjPix	Counts	Number of source-object-coverage pixels.
7	nNanPix	Counts	Number of NaN (not a number) pixels.
8	nDirtPix	Counts	Number of pixels with filter dirt.
9	nStarPix	Counts	Number of star-coverage pixels.
10	nGalxPix	Counts	Number of galaxy-coverage pixels.
11	nObjSex	Counts	Number of source objects found by SExtractor.
12	fwhmSex	Arcseconds	SExtractor FWHM of the radial profile.
13	gMean	D.N.	Image global mean.
14	gMedian	D.N.	Image global median.
15	cMedian1	D.N.	Image upper-left corner median.
16	cMedian2	D.N.	Image upper-right corner median.
17	cMedian3	D.N.	Image lower-right corner median.
18	cMedian4	D.N.	Image lower-left corner median.
19	gMode	D.N.	Image global mode.
20	MmFlag	Counts	Image global mode.
21	gStdDev	D.N.	Image global standard deviation.
22	gMAbsDev	D.N.	Image mean absolute deviation.
23	gSkewns	D.N.	Image skewness.
24	gKurtos	D.N.	Image kurtosis.
25	gMinVal	D.N.	Image minimum value.
26	gMaxVal	D.N.	Image maximum value.
27	pTile1	D.N.	Image 1 percentile.
28	pTile16	D.N.	Image 16 percentile.
29	pTile84	D.N.	Image 84 percentile.
30	pTile99	D.N.	Image 99 percentile.
31	photCalFlag	Flag	Flag for whether image could be photometrically calibrated.
32	zeroPoint	Magnitudes	Magnitude zero point at an air mass of zero (see Appendix).
33	extinction	Magnitudes	Extinction.
34	airMass	None	Air mass.
35	photCalChi2	None	Chi2 of photometric calibration.
36	photCalNDegFreedom	Counts	Number of SDSS matches in photometric calibration.
37	photCalRMSE	Magnitudes	R.M.S.E. of photometric calibration.
38	aveDeltaMag	Magnitudes	Average delta magnitude over SDSS sources in a given image.
40	nPhotSources	Counts	Number of sources used in photometry calibration.
41	astrrms1	Degrees	SCAMP astrometry rms along axis 1 (ref., high signal-to-noise ratio [S/N]).
42	astrrms2	Degrees	SCAMP astrometry rms along axis 2 (ref., high S/N).
43	2mass_astrrms1	Arcseconds	2Mass astrometry rms along axis 1.
44	2mass_astrrms2	Arcseconds	2Mass astrometry rms along axis 2.
45	2mass_astravg1	Arcseconds	2Mass astrometry match-distance average along axis 1.
46	2mass_astravg2	Arcseconds	2Mass astrometry match-distance average along axis 2.
47	n2massMatches	Counts	Number of 2MASS sources matched.
48	imageZeroPoint	Magnitudes	Magnitude zero point of image determined directly from SDSS sources (see Appendix).
49	imageColorTerm	Magnitudes	Color term from data-fit to SDSS sources in a given image (see Appendix).
50	2mass_astrrms1_11	Arcseconds	2MASS astrometry rms along axis 1 for subimage (1, 1).
51	2mass_astrrms2_11	Arcseconds	2MASS astrometry rms along axis 2 for subimage (1, 1).
52	2mass_astravg1_11	Arcseconds	2MASS astrometry match-distance average along axis 1 for subimage (1, 1).
53	2mass_astravg2_11	Arcseconds	2MASS astrometry match-distance average along axis 2 for subimage (1, 1).
54	n2massMatches_11	Counts	Number of 2MASS sources matched for subimage (1, 1).
55	2mass_astrrms1_12	Arcseconds	2MASS astrometry rms along axis 1 for subimage (1, 2).
56	2mass_astrrms2_12	Arcseconds	2MASS astrometry rms along axis 2 for subimage (1, 2).
57	2mass_astravg1_12	Arcseconds	2MASS astrometry match-distance average along axis 1 for subimage (1, 2).
58	2mass_astravg2_12	Arcseconds	2MASS astrometry match-distance average along axis 2 for subimage (1, 2).
59	n2massMatches_12	Counts	Number of 2MASS sources matched for subimage (1, 2).
60	2mass_astrrms1_13	Arcseconds	2MASS astrometry rms along axis 1 for subimage (1, 3).

NOTE.—For the SDQA metrics associated with subimages, the size for subimages (1, j) and (3, j) is 768×1024 pixels, and the size for subimages (2, j) is 768×2048 pixels.

TABLE 8

IPAC-PTF SDQA METRICS STORED IN THE *SDQA_METRICS* DATABASE TABLE (CONTINUED FROM TABLE 7)

metricId	metricName	physicalUnits	Definition
61	2mass_astrrms2_13	Arcseconds	2MASS astrometry rms along axis 2 for subimage (1, 3).
62	2mass_astravg1_13	Arcseconds	2MASS astrometry match-distance average along axis 1 for subimage (1, 3).
63	2mass_astravg2_13	Arcseconds	2MASS astrometry match-distance average along axis 2 for subimage (1, 3).
64	n2massMatches_13	Counts	Number of 2MASS sources matched for subimage (1, 3).
65	2mass_astrrms1_21	Arcseconds	2MASS astrometry rms along axis 1 for subimage (2, 1).
66	2mass_astrrms2_21	Arcseconds	2MASS astrometry rms along axis 2 for subimage (2, 1).
67	2mass_astravg1_21	Arcseconds	2MASS astrometry match-distance average along axis 1 for subimage (2, 1).
68	2mass_astravg2_21	Arcseconds	2MASS astrometry match-distance average along axis 2 for subimage (2, 1).
69	n2massMatches_21	Counts	Number of 2MASS sources matched for subimage (2, 1).
70	2mass_astrrms1_22	Arcseconds	2MASS astrometry rms along axis 1 for subimage (2, 2).
71	2mass_astrrms2_22	Arcseconds	2MASS astrometry rms along axis 2 for subimage (2, 2).
72	2mass_astravg1_22	Arcseconds	2MASS astrometry match-distance average along axis 1 for subimage (2, 2).
73	2mass_astravg2_22	Arcseconds	2MASS astrometry match-distance average along axis 2 for subimage (2, 2).
74	n2massMatches_22	Counts	Number of 2MASS sources matched for subimage (2, 2).
75	2mass_astrrms1_23	Arcseconds	2MASS astrometry rms along axis 1 for subimage (2, 3).
76	2mass_astrrms2_23	Arcseconds	2MASS astrometry rms along axis 2 for subimage (2, 3).
77	2mass_astravg1_23	Arcseconds	2MASS astrometry match-distance average along axis 1 for sub-image (2, 3).
78	2mass_astravg2_23	Arcseconds	2MASS astrometry match-distance average along axis 2 for subimage (2, 3).
79	n2massMatches_23	Counts	Number of 2MASS sources matched for subimage (2, 3).
80	2mass_astrrms1_31	Arcseconds	2MASS astrometry rms along axis 1 for subimage (3, 1).
81	2mass_astrrms2_31	Arcseconds	2MASS astrometry rms along axis 2 for subimage (3, 1).
82	2mass_astravg1_31	Arcseconds	2MASS astrometry match-distance average along axis 1 for subimage (3, 1).
83	2mass_astravg2_31	Arcseconds	2MASS astrometry match-distance average along axis 2 for subimage (3, 1).
84	n2massMatches_31	Counts	Number of 2MASS sources matched for subimage (3, 1).
85	2mass_astrrms1_32	Arcseconds	2MASS astrometry rms along axis 1 for subimage (3, 2).
86	2mass_astrrms2_32	Arcseconds	2MASS astrometry rms along axis 2 for subimage (3, 2).
87	2mass_astravg1_32	Arcseconds	2MASS astrometry match-distance average along axis 1 for subimage (3, 2).
88	2mass_astravg2_32	Arcseconds	2MASS astrometry match-distance average along axis 2 for subimage (3, 2).
89	n2massMatches_32	Counts	Number of 2MASS sources matched for subimage (3, 2).
90	2mass_astrrms1_33	Arcseconds	2MASS astrometry rms along axis 1 for subimage (3, 3).
91	2mass_astrrms2_33	Arcseconds	2MASS astrometry rms along axis 2 for subimage (3, 3).
92	2mass_astravg1_33	Arcseconds	2MASS astrometry match-distance average along axis 1 for subimage (3, 3).
93	2mass_astravg2_33	Arcseconds	2MASS astrometry match-distance average along axis 2 for subimage (3, 3).
94	n2massMatches_33	Counts	Number of 2MASS sources matched for subimage (3, 3).
95	medianSkyMag	Magnitudes $(\text{s arcsec}^2)^{-1}$	Median sky magnitude.
96	limitMag	Magnitudes $(\text{s arcsec}^2)^{-1}$	Limiting magnitude (obsolete method).
97	medianFwhm	Arcseconds	Median FWHM.
98	medianElongation	None	Median elongation.
99	stdDevElongation	None	Standard deviation of elongation.
100	medianTheta	Degrees	Special median of THETAWIN_WORLD.
101	stdDevTheta	Degrees	Special standard deviation of THETAWIN_WORLD.
102	medianDeltaMag	Magnitudes $(\text{s arcsec}^2)^{-1}$	Median (MU_MAX −MAG_AUTO).
103	stdDevDeltaMag	Magnitudes $(\text{s arcsec}^2)^{-1}$	Std. dev of (MU_MAX − MAG_AUTO).
104	scampCatType	None	SCAMP-catalog type: 1=SDSS-DR7, 2=UCAC3, 3=USNO-B1
105	nScampLoadedStars	None	Number of stars loaded from SCAMP input catalog.
106	nScampDetectedStars	None	Number of stars detected by SCAMP.
107	imageZeroPointSigma	Magnitudes	Sigma of magnitude difference between SExtractor and SDSS sources.
108	limitMagAbsPhotCal	Magnitudes $(\text{s arcsec}^2)^{-1}$	Limiting magnitude (abs. phot. cal. zero point).
109	medianSkyMagAbsPhotCal	Magnitudes $(\text{s arcsec}^2)^{-1}$	Median sky magnitude based on abs. phot. cal. zero point.
110	flatJarqueBera	Dimensionless	Jarque-Bera test for abnormal data distribution of superflat image.
111	flatMean	Dimensionless	Mean of superflat image.
112	flatMedian	Dimensionless	Median of superflat image.
113	flatStdDev	Dimensionless	Standard deviation of superflat image.
114	flatSkew	Dimensionless	Skew of superflat image.
115	flatKurtosis	Dimensionless	Kurtosis of superflat image.
116	flatPercentile84.1	Dimensionless	84.1 percentile of superflat image.
117	flatPercentile15.9	Dimensionless	15.9 percentile of superflat image.
118	flatScale	Dimensionless	Scale (one half the difference between 84.1 and P15.9 percentiles) of superflat image.
119	flatNumNanPix	Counts	Number of NaNed pixels in superflat image.

NOTE.—For the SDQA metrics associated with subimages, the size for subimages (1, j) and (3, j) is 768×1024 pixels, and the size for subimages (2, j) is 768×2048 pixels.

can include lower and/or upper thresholds. Since thresholds can change over time as the SDQA subsystem is tuned, the table has *version* and *vBest* columns to keep track of the different and best versions (like the *ProcImages* database table).

The *SDQA_Ratings* database table is associated with the *ProcImages* database table in a one-to-many relationship record-wise, and, for a given processed image, stores multiple records of what we refer to as image "SDQA ratings," which are the values associated with SDQA metrics (referred to above). An SDQA rating is basically the computed value of an SDQA metric and its uncertainty. This design encourages the storing of an uncertainty with its computed SDQA-rating value, although this is not required. The *flagValue* column in a given record is normally set to zero, but is reset to one when the associated *metricValue* falls outside of the region allowed by the corresponding threshold(s). A processed image, in general, has many different SDQA ratings, as noted above, which are computed at various pipeline stages; PTF processed images each have over 100 different SDQA ratings (see Tables 7 through 8). An *SDQA_Ratings* record contains indexes to the relevant processed image, SDQA metric, and SDQA threshold, which are foreign keys. The *SDQA_Ratings* database table potentially will have a large number of records; bulk loading of these records may reduce the impact of the SDQA subsystem on pipeline throughput, although this has not been necessary for IPAC-PTF pipelines.

A separate database-stored function called *setSdqaStatus (pid)* is called to compute the SDQA grade of a processed image after its SDQA ratings have been loaded into the database. The function computes the percentage of SDQA ratings that are flagged ($flagValue = 1$ in the *SDQA_Ratings* database table). The possible pipeline-assigned SDQA status values are listed in Table 9.

9. IMAGE-PROCESSING PIPELINES

9.1. Overview

The pipelines consist of Perl scripts and the modules or binary executables that they run. The modules are either custom developed in house or freely downloadable astronomical-software packages (e.g., SExtractor). There are product-generation and calibration pipelines (see Table 10), which must be executed in a particular order.

In normal operations, the pipelines are initiated via multi-threaded job client software developed expressly for PTF at IPAC. One job client is typically run on one pipeline machine at any given time. The job clients interact with the database to coordinate the pipeline jobs. The database maintains a queue of jobs waiting to be processed. Each job is associated with a particular pipeline and data set. Job clients that are not busy periodically poll the database for more jobs, which responds with the database identifications of jobs to process, along with concise information about the jobs that is needed by the pipelines. The job client then launches the called-for pipeline as a separate processing thread and is typically blocked until the thread completes. The database is updated with relevant job information after the job finishes (e.g., pipeline start and end times).

The pipelines nominally query the database for any additional metadata that are required to run the pipeline. The last step of the pipeline includes updating the database with metadata about the processed-image product(s) and their ancillary files (e.g., data masks). The pipelines make and sever database connections as needed, and database communications to the pipeline and to the job executive are independent.

The pipelines create numerous intermediate data files on the pipeline machine's local disk, which are handy to have for manually rerunning pipeline steps, should the need arise. A fraction of these files are copied to a sandbox disk (see § 5), which serves to marshal together the products for a given night generated in parallel on different pipeline machines. It is expedient to organize the products in the sandbox in subdirectories that make them easy to find without having to query the database. The following sample file path exemplifies the subdirectory scheme that we have adopted:

```
/sbx1/2011/09/19/f2/c9/p5/v1.
```

TABLE 9
POSSIBLE SDQA STATUS VALUES

sdqa_statusid	statusName	SDQA ratings flagged (%)	Definition
1	passedAuto	<5	Image passed by automated SDQA.
2	marginallyPassedAuto	≥ 5 and <25	Image marginally passed by automated SDQA.
3	marginallyFailedAuto	>75	Image marginally failed by automated SDQA.
4	failedAuto	≥ 90	Image failed by automated SDQA.
5	indeterminateAuto	≥ 25 and ≤ 75	Image is indeterminate by automated SDQA.
6	passedManual	N/A	Image passed by manual SDQA.
7	marginallyPassedManual	N/A	Image marginally passed by manual SDQA.
8	marginallyFailedManual	N/A	Image marginally failed by manual SDQA.
9	failedManual	N/A	Image failed by manual SDQA.
10	indeterminateManual	N/A	Image is indeterminate by manual SDQA.

TABLE 10

CONTENTS OF THE *PIPELINES* DATABASE TABLE

ppid[a]	Priority[b]	Blocking	Perl script	Description
1	10	1	*superbias.pl*	Superbias calibration
2	20	1	*domeflat.pl*	Dome flat calibration
3	30	1	*preproc.pl*	Raw-image preprocessing
4	40	1	*superflat.pl*	Superflat calibration
5	50	1	*frameproc.pl*	Frame processing
6	70	1	TBD	Mosaicking
7	500	1	*splitCameraImages.pl*	Camera-image splitting
8	60	1	*sourceAssociation.pl*	Source association
9	55	0	*loadSources.pl*	Load sources into database
10	45	1	*flattener.pl*	Flattener
11	41	1	*twilightflat.pl*	Twilight flat
12	80	1	*genRefImage.pl*	Reference image
13	52	1	*genCatalog.pl*	Source-catalog generation

[a] Pipeline database index.

[b] The priority numbers are relative, and smaller numbers have higher priority.

After the sandbox logical name and the year, month, and day, there is "f2/c9/p5/v1," which stands for filter ($fid = 2$), CCD ($ccdid = 9$), pipeline ($ppid = 5$), and product version ($version = 1$). The directory tree for the archive is exactly the same, except that the archive logical name replaces the sandbox's. The method employed for copying products from the sandbox to the archive is described in § 10.1.

9.2. Computing Environment

The pipelines inherit the shell environment they run under, which is overridden by settings particular to the PTF software system (see Table 11). A modest number of environment variables is required. The *PATH* environment variable must include locations of PTF scripts and binary executables, Perl, Python,

TABLE 11

ENVIRONMENT VARIABLES REQUIRED BY THE PTF SOFTWARE SYSTEM

Variable	Definition
PTF_ROOT	Root directory of PTF software system.
PTF_LOGS	Directory of log files (e.g., *$PTF_ROOT/logs*).
PTF_ARCHIVE	Archive directory (e.g., *$PTF_ROOT/archive*).
PTF_ARCHIVE_RAW_PARTITION	Archive raw-data disk partition (e.g., *raw*).
PTF_ARCHIVE_PROC_PARTITION	Archive processed-data disk partition (e.g., *proc*).
PTF_SBX	Current sandbox directory (e.g., *$PTF_ROOT/sbx1*).
PTF_SW	Top-level software directory (e.g., *$PTF_ROOT/sw*).
PTF_BIN	Binary-executables directory (e.g., *$PTF_SW/ptf/bin*).
PTF_LIB	Libraries directory (e.g., *$PTF_SW/ptf/lib*).
PTF_EXT	External-software directory (e.g., *$PTF_ROOT/ext*).
PTF_LOCAL	Machine local directory (e.g., */scr/ptf*).
PTF_CDF	Configuration-data-file directory (e.g., */scr/cdf*).
PTF_CAL	Calibration-file directory (e.g., */scr/cal*).
PTF_IDL	Full path and filename of IDL program.
PTF_ASTRONOMYNETBIN	Astrometry.net binary-executable directory.
WRAPPER_UTILS	Perl-library directory (e.g., $*PTF_SW*/perlibs).
WRAPPER_VERBOSE	Pipeline verbosity flag (0 or 1).
DBTYPE	Database type.
DNAME	Database name.
DBSERVER	Database-server name.
SODB_ROLE	Database role.
TY2_PATH	Location of the Tycho-2 catalog.
PATH	Location(s) of binary executables (e.g., $*PTF_BIN*).
LD_LIBRARY_PATH	Location(s) of libraries (e.g., $*PTF_LIB*).
PERL_PATH	Location of Perl-interpreter command.
PERL5LIB	Location(s) of Perl-library modules.
PYTHONPATH	Location of Python-interpreter command.

TABLE 12
VERSIONS OF THIRD-PARTY SOFTWARE EXECUTED IN IPAC-PTF PIPELINES

Software	Version
Astrometry.net	0.43
CFITSIO	3.35
Eye	1.3.0
FFTW	3.2.2
IDL	8.1
MATLAB	7.10.0.499
Montage	3.2
Perl	5.10.0
Python	2.7.3
EPD	7.3-2
SCAMP	1.7.0
MissFITS	2.4.0
SExtractor	2.8.6
SWarp	2.19.1
WCSTools	3.8.7
DAOPHOT	2004 Jan 15
ALLSTAR	2001 Feb 7
SciApps	08/29/2011

MATLAB, Astrometry.net, and Jessica Mink's WCSTools. The *PTF_IDL* environment variable gives the path and command name of IPAC's SciApps installation of IDL. Table 12 lists the versions of third-party software utilized in IPAC-PTF pipelines.

9.3. Configuration Data Files

Configuration data files (CDFs) are text files that store configuration data in the form of keyword=value pairs. They are parameter files that control software behavior. On the order of a hundred of these files are required for PTF processing. In many cases, there are sets of 11 files for a given process working on individual CCDs, thus allowing CCD-dependent image processing. The CDFs for the superbias-calibration pipeline (see § 9.11), for example, store the outlier-rejection threshold and the pixel coordinates of the floating-bias strip. Among the files are SExtractor "config" and "param" files. The CDFs are version-controlled in CVS, and the version numbers of the CDFs as a complete set of files are tracked in the *CdfVersions* database table, along with deployment dates and times, etc. For fast access, the CDFs are stored locally on each pipeline machine's scratch disk (as defined by environment variable *PTF_CDF*; see § 9.2).

9.4. Pixel-Mask Images

Pixel masks are used to flag any badly behaved pixels on the CCDs. The flagged pixels can be specially treated by the image-processing pipelines, as appropriate. The pixel masks for PTF data were constructed as described by van Eyken et al. (2011). The algorithm is loosely based on the IRAF[17] *ccdmask* procedure (Tody 1986, 1993). The masks were created from images made by dividing a 70 s LED[18] flat field by a 35 s LED flat field. Three independent such divided frames were obtained for each of the 11 functioning CCDs. Any pixels with outlier fluxes beyond four standard deviations in at least two of the three frames, or beyond three standard deviations in all three of the frames were flagged as bad. This approach helps catch excessively variable pixels, in addition to highly nonlinear pixels, while still rejecting cosmic-ray hits. The bad-pixel-detection procedure was then repeated after boxcar smoothing of the original image along the readout direction. This finds column segments where individual pixels are not statistically bad when considered alone, but are statistically bad when taken together as an aggregate. This process was iterated several times, with a selection of smoothing bin sizes from 2 to 20 pixels. Pixels lying in small gaps between bad pixels were then also iteratively flagged, with the aim of completely blocking out large regions of bad pixels while minimizing encroachment into good-pixel regions.

9.5. Pipeline Executive

The pipeline executive is software that runs in parallel on the pipeline machines as pipeline job clients. There is no pipeline-executive server per se, as its function has been replaced by a relational database. The pipeline executive expects pipeline jobs to be inserted as records in the *Jobs* database table, which is an integral part of the operations database schema (see § 6). Thus, staging pipeline jobs for execution is as simple as inserting database records and assuring that the records are in the required state for acceptance by the pipeline executive. The *Jobs* database table is queried for a job when a pipeline machine is not currently running a job and its job client is seeking a new job. The job farmed out to a machine will be next in the priority ordering, which is specified in the *Pipelines* database table. The current contents of this table are listed in Table 10. The pipeline-priority numbers are relative and can be renumbered as new pipelines are added or priority changes arise.

A *Jobs* database record is prepared for pipeline running by nulling out the run-time columns and setting the status to zero. Staged jobs that have not yet been executed can be suspended by setting their status to -1 and then reactivated later by setting their status back to zero.

The job-client software is written in Perl (*ptfJobber.pl*) and has an internal table that associates each of the 11 PTF CCDs with a different pipeline machine. It allows a pipeline machine to either run only jobs for the associated CCD or jobs that are CCD independent (e.g., the camera-image-splitting pipeline described in § 9.10). It runs in an open loop, and wakes up every 5 s to check whether a job has completed and/or a new job can be started.

[17] http://iraf.noao.edu/.

[18] Light-emitting diode; see Law et al. (2009).

Each client maintains a list of launched pipelines that grows indefinitely (until stopped and restarted, which, for example, is done for the weekly database backup). Each launched pipeline executes as a separate processing thread. The attributes of the launched pipelines include their job database identifications (*jid*), whether the job has completed, and whether the job is nonblocking ($blocking = 0$; see Table 10). If the job currently being run by the client has a pipeline-blocking flag of one, then the client will wait for the job to finish before requesting another job. If, on the other hand, the job is nonblocking, then the client will request another job and run it in parallel to the first job as another processing thread. The client is currently limited to running only one nonblocking job in parallel to a blocking job, but this can be increased by simply changing a parameter.

9.6. Virtual Pipeline Operator

Running pipelines and archiving the products, delivering product metadata to IRSA, and other routine daily operations are automated with a Perl script that we call the virtual pipeline operator (VPO). In addition, the script monitors disk usage, sends e-mail notifications and nightly summaries, and runs a nightly process that generates all-sky depth-of-coverage images (Aitoff projections in Galactic and equatorial coordinates).

The VPO can be run in open-loop mode for continuous operation. The polling-time interval is currently set at 10 minutes. The software can also be run in single-night mode for targeted reprocessing. It does much of its work by querying the database for information, and, in particular, the *Jobs* database table for pipeline monitoring. It is basically a finite state machine that sets internal flags to keep track of what has been done and what needs to be done still for a given night's worth of data. The flags are also written to a state file, which is unique for a given night, each time the state is updated. The software is easily extensible by a Perl programmer when additional states and/or tasks are needed. It resets to default initial-state values every 24 hr; currently this is set to occur at 10 A.M., which is around the time the data-ingestion process completes for the previous night and its pipeline processing can be started.

The VPO can also read the initial state from a hand-edited input file (preferably by an expert pipeline operator). This is advantageous when an error occurs and the VPO must be restarted at some intermediate point. There are combinations of states that are not allowed, and the software could be made more robust by adding checks for invalid states.

9.7. Archival Filenames

Pipeline-product files are created with fixed, descriptive filenames (e.g., "superflat.fits"), and then renamed to have unique filenames near the end of the pipeline. The unique filenames are of constant length and have 11 identifying fields arranged in a standardized form. Table 13 defines the 11 fields and gives an example filename. The filename fields are delimited by an underscore character and are all lowercase, except for the first field. If necessary, a filename field is padded with leading zeros to keep the filename length constant. The filename contains enough information to identify the file precisely.

The structure of the archive directory tree, in which the archived products are stored on disk, has already been described in § 9.1.

9.8. Pipeline Multithreading

Parallel image-processing on each of our pipeline machines is possible, given the machine architecture (see § 5), and this is

TABLE 13

STANDARDIZED FILE-NAMING SCHEME FOR PTF PRODUCTS

Filename field #[a]	Definition
1	Always "PTF" (uppercase)
2	Concatenation of year (4 digits), month (2 digits), day (2 digits), and fractional day (4 digits)
3	One-character product format[b]
4	One-character product category[c]
5	Four-character product type[d]
6	Prefix "t" for time followed by hours (2 digits), minutes (2 digits), and seconds (2 digits)
7	Prefix "u" for unique index followed by relevant database-table primary key
8	Prefix "f" for filter followed by 2-digit filter number (*FILTERID*)
9	Prefix "p" for PTF field followed by PTF field number (*PTFFIELD*)
10	Prefix "c" for CCD followed by two-digit CCD index (*CCDID*)
11	Filename extension (e.g., "fits" or "ctlg")

[a] Sample filename: PTF_200903011372_i_p_scie_t031734_u008648839_f02_p000642_c10.fits.

[b] Choice of "i" for image or "c" for catalog.

[c] Choice of "p" for processed, "s" for super, or "e" for external.

[d] Choice of "scie" for science, "mask" for mask, "bias" for superbias, "banc" for superbias-ancillary file, "flat" for superflat, "twfl" for twilight flat, "fmsk" for flat mask, "weig" for weight, "zpvm" for zero-point variability map, "zpve" for zero-point-variability-map error, "sdss" for SDSS, "uca3" for UCAC3, "2mas" for 2MASS (Two-Micron All-Sky Survey), or "usb1" for USNO-B1.

enabled in our pipelines by the Perl *threads* module. Some modules executed by our pipelines, such as SCAMP (Bertin 2006b) and SExtractor (Bertin & Arnouts 1996), are also multithreaded codes, and the maximum number of threads they run simultaneously must be limited when running multiple threads at the Perl-script level.

Our pipelines currently run only a single instance of the astrometry-refinement code, SCAMP, at a time and in a configuration that will cause it to automatically use as many threads as there are cores in the machine (which is eight). The pipelines run multithreaded SExtractor built to allow up to two threads and let the Perl wrapper code control the multithreading at a higher level.

The multithreading in the Perl pipeline scripts is nominally configured to allow up to seven threads at a time, which we found is optimal for nonthreaded parallel processes through benchmark testing on our pipeline machines. Wherever in our pipelines running a module in multithreaded mode is determined to be advantageous, a master thread is launched to oversee the multithreaded processing for the module, and then are launched multiple slave threads running separate instances of the module on different images or input files in parallel. For thread synchronization, a thread-join function is called to wait for all threads to complete before moving on to the next step in the pipeline. The exit code from each thread is checked for abnormal termination.

9.9. Stand-Alone Pipeline Execution

PTF pipelines can be easily executed outside of the pipeline executive. Since the pipelines query a database for inputs, the particular database used must be updated with pointers to the input files on disk. Once the raw data for a given night have been ingested, the database is updated automatically as the pipelines are run in proper priority order (see Table 10).

The simplicity of the basic instructions for standalone pipeline execution are illustrated in the following example, in which the superbias pipeline is executed:

```
cd /scr/work/dir
source $PTF_SW/ptf/ops/ops.env
setenv PTF_SBX /user/sbx1
setenv DBNAME user22
setenv DBSERVER dbsvr42
setenv PIPEID 1
setenv RID 34
$PTF_SW/ptf/src/pl/perl/superbias.pl.
```

The selected working directory serves the same purpose as the pipeline machine's local disk where all pipeline intermediate data files are written. Stand-alone pipeline execution is therefore useful for diagnosing problems. After sourcing the basic environment file, generally the user will want to override the environment variables that point to the user's sandbox and database. The user's database is normally a copy of the operations database. Environment variables *RID*, which is a representative raw-image database identification (*rid*), and *PIPEID*, which is the pipeline database ID (*ppid*), reference the input data and pipeline number to be executed, respectively. In this particular case, the representative image is representative of all bias images taken for a given night and CCD; in the case of the superflat pipeline, the representative image is representative of all science images (i.e., *IMGTYP* = "object") for a given night, CCD, and filter. Once the pipeline is set up using these commands, the pipeline is executed with the last command listed above. In most cases, the user will want to redirect the standard output and error streams to a log file. The basic procedure is similar for all PTF pipelines and can easily be scripted if a large number of pipeline instances are involved.

9.10. Camera-Image-Splitting Pipeline

After the PTF data for a given night are ingested, the camera-image-splitting pipelines, one pipeline instance per camera exposure, are launched automatically by the high-level data-ingest process (see § 7.1), or by the VPO (see § 9.6) in the case that the data had to be manually ingested because of some abnormal condition. The pipeline executive is set up to execute one instance of this pipeline per machine at a time. Since there are 11 pipeline machines, 11 instances of the pipeline are run in parallel. This particular pipeline is not particularly compute or memory intensive, and so more of these pipeline instances per machine could be run, and tests of up to four instances per machine have been performed successfully.

The camera-image-splitting pipeline is wrapped in a Perl script called *splitCameraImages.pl*. The input camera-image file is copied from the archive to the pipeline machine's scratch disk. The checksum of the file is recomputed and compared to the checksum stored in the database, and a mismatch, like any other pipeline error, would result in a diagnostic message written to the log file and pipeline termination with exit code $>= 64$. The filter associated with the camera-image file is verified by running *check_filter.py*, which uses median values of various regions of image data and smoothing to look for patterns in the data that have high amplitude for the g band but are weak for the R band. A filter mismatch results in pipeline termination with exit code=65. Manual intervention is required in this case to decide whether to alter the filter information in the database (filter-changer malfunctions have occurred intermittently during the project) or skip the filter checking for that pipeline. Experience has shown that this filter checking is not reliable when the seeing is poor.

The module *ptfSplitMultiFITS* is executed on the camera-image file to break it up into 12 single-extension FITS files. The primary HDU, plus CCD-dependent keywords for the gain, read noise, and dark current (*GAIN*, *READNOI*, and *DARKCUR*, respectively) are copied to the headers of the split-up files. The

resulting single CCD-image FITS files are then processed separately (except for dead $CCDID = 3$, which is skipped).

If the CCD images are science images ($itid = 1$; see Table 6), then they are processed to find first-iteration astrometric solutions. Initial values of world-coordinate-system (WCS) keywords are written to the CCD-image FITS headers. *CRVAL1* and *CRVAL2*, the coordinates of the WCS reference point on the sky, are set to the right ascension and declination of the telescope boresight, *TELRA* and *TELDEC*, respectively. *CRPIX1* and *CRPIX2*, the corresponding reference-point image coordinates for a given CCD, are set to the telescope-boresight pixel positions that have been predetermined for each CCD-image reference frame. Finally, the following fixed values for the pixel scale (at the distortion center) and image rotation angle are set, as appropriate for the telescope and camera: $CDELT1 = -0.000281°$, $CDELT2 = 0.000281°$, and $CROTA2 = 180°$. Next, source extraction is done with SExtractor (Bertin & Arnouts 1996; Bertin 2006a; Holwerda 2005) to generate a source catalog for the astrometry. The pipeline then runs Astrometry.net modules *augment-xylist*, *backend*, and *new-wcs* (Lang et al. 2010) in succession with the objective of finding an astrometric solution.

If an astrometric solution is found, then it is verified and recorded. Verification includes requiring the pixel scale to be within $\pm 5\%$ of the initial known value, the rotation angle to be within 5° of the initial known value, and the absolute values of *CRPIX1* and *CRPIX2* to be $\leq 10,000$ pixels. If these conditions are not met, then bit $2^3 = 8$ is set in the *infobits* column of the *RawImages* database table (see Table 14) to flag this condition. The astrometric solution is written both to the FITS header of the CCD image and also to a text file in the archive containing only the astrometric solution, in order to facilitate later generation by IRSA of source-catalog overlays onto JPEG preview images of PTF data.

The CCD-image files are copied to the sandbox into a hierarchical directory tree that differentiates the stored files by observation year, month, day, filter identification, CCD identification, and pipeline database identification. A record is created in the *RawImages* database table for each CCD-image file. The record contains a number of useful foreign keys to other database tables (*expid*, *ccdid*, *nid*, *itid*, *piid*) and comprises columns for storing the location and name of the file, record-creation date, image status, checksum, and *infobits*. The image status can be either zero or one, and is normally zero only for the dead CCD ($CCDID = 3$). A bad astrometric solution, although flagged in the *infobits* column of the *RawImages* database table, will not result in $status = 0$ for the image at this point because the downstream frame-processing pipeline (see § 9.15) will make another attempt at finding a good solution.

The pipeline makes preview images in JPEG format using IRSA's Montage software, both for the camera 12-CCD-composite image and individual CCD images. The preview images are subsequently used by the SDQA subsystem (see § 8).

TABLE 14

BITS ALLOCATED FOR FLAGGING VARIOUS CONDITIONS AND EXCEPTIONS IN THE *INFOBITS* COLUMN OF THE *RAWIMAGES* DATABASE TABLE

Bit		Definition
0	.....	Dead CCD
1	.....	Astrometry.net failed
2	.....	Sidereal-tracking failure[a]
3	.....	Bad astrometric solution
4	.....	Transient noise in image[a]

[a] Manually set after image inspection.

9.11. Superbias-Calibration Pipeline

The purpose of the superbias calibration pipeline is to compute the pixel-by-pixel electronic bias correction that is applied to every PTF science image. These pipelines are launched after the camera-image-splitting pipelines have completed for a given night, one pipeline instance per CCD per night. This is done either automatically by the VPO or manually by a human pipeline operator.

The superbias pipeline is wrapped in a Perl script called *superbias.pl*. The database is queried for all bias images for the night and CCD of interest. The *ptfSuperbias* module is then executed, and this produces the superbias-image calibration file, a file called "superbias.fits," which is the common bias in the image data for a given CCD and night. The file is renamed to an archival filename, copied to the sandbox, and registered in the *CalFiles* database table with *caltype* = "superbias."

The method used to compute the superbias is described as follows: The bias images are read into memory. The floating bias of each image is computed and then subtracted from its respective bias image. The CCD-appropriate pixel mask is used to ignore dead or bad pixels. The software can be set up to compute the floating bias from up to three different overscan regions, but, in practice, only the long strip running down the right-hand side of the image is utilized. The floating bias is the average of the values in the overscan region after an aggressive outlier-rejection step. The outliers are found by thresholding the data at the median value ± 2.5 times the data dispersion, which is given by half of the difference between the 84.1 percentile and the 15.9 percentile. The bias-minus-floating-bias values are then processed by a similar outlier-rejection algorithm on a pixel-by-pixel basis, and the surviving values are averaged at each pixel location to yield the superbias image and accompanying ancillary images, which are described in the next paragraph.

Ancillary calibration products are also generated by the *ptfSuperbias* module. These are packed into a file called "superbias_ancil_data.fits." The ancillary FITS file is an image-data cube ($NAXIS = 3$) containing the superbias uncertainties in the first data plane, the number of samples in the second data plane, and the number of outliers rejected in the third data plane.

All quantities are on a pixel-by-pixel basis. The file is renamed to an archival filename, copied to the sandbox, and registered in the *AncilCalFiles* database table with *anciltype* = "superbiasstats."

9.12. Preprocessing Pipeline

The preprocessing pipeline prepares the science images (*IMGTYP* = "object") to be fed into the downstream superflat-calibration and image-flattener pipelines. The preprocessing is severalfold:

1. Subtract off the floating bias and superbias from each pixel value;
2. Crop the science images to remove the bias overscan regions;
3. Compute data-mask bit settings for saturated and "dirty" pixels (bit $2^8 = 256$ and bit $2^{11} = 2048$, respectively; see Table 15; "dirty" pixels are defined below), and combine them with the appropriate fixed, CCD-dependent pixel mask (see § 9.4) to create an initial data mask for every science image;
4. Recompute an improved value for the seeing; and
5. Augment the data-mask image for each science image with the bit setting allocated for marking object detections (bit $2^1 = 2$; see Table 15) taken from SExtractor object check images.

The preprocessing pipeline is wrapped in a Perl script called *preproc.pl*. An instance of this pipeline runs on a per-night, per-CCD, per-filter basis. The saturation level for the CCD at hand is looked up at the beginning of the pipeline.

The preprocessing pipeline requires the following input calibration files: a pixel mask and a superbias image. It will also utilize a superflat image, if available. The calibration files are retrieved via a call to database stored function *getCalFiles*, which queries the *CalFiles* database table, and returns a hash table of the latest calibration files available for the night, CCD, and filter of interest. The function always returns fallback calibration files for the superbias and superflat, which are zero-value and unity-value images, respectively. The fallbacks are pressed into service when the primary calibration files are nonexistent.

The bit allocations for data-mask images are documented in Table 15. Bit $2^1 = 2$ is allocated for pixels overlapping onto detected astronomical objects. Bit $2^8 = 256$ is allocated for saturated pixels. Bit $2^{11} = 2048$ is allocated for dirty pixels, where "dirty" is defined as 10 standard deviations below the image's local median value.

The pipeline first runs the *ptfSciencePipeline* module to perform bias corrections, image cropping, and computation of the initial data masks. The floating bias is computed via the method described above (see § 9.11). The pipeline runs multiple threads of this process, where each thread processes a portion of the input science images in parallel. The science images are cropped to 2048×4096 pixels. The pipeline outputs are a set of bias-corrected images and a set of bias-corrected and flattened images (useful if a flat happens to be available from a prior run).

Next, multithreaded runs of SExtractor are made on the aforementioned latter set of images, one thread per image, in order to generate source catalogs for the seeing calculation. Object check images are also generated in the process. Bit $2^7 = 128$ will be set in the *infobits* column of the *ProcImages* database table (see Table 16) for $ppid = 3$ records associated with science images that contain no sources.

TABLE 15

BITS ALLOCATED FOR DATA MASKS

Bit	Definition
0	Aircraft/satellite track
1	Object detected
2	High dark current
3	Reserved
4	Noisy
5	Ghost
6	CCD bleed
7	Radiation hit
8	Saturated
9	Dead/bad
10	NaN (not a number)
11	Dirt on optics
12	Halo
13	Reserved
14	Reserved
15	Reserved

TABLE 16

BITS ALLOCATED FOR FLAGGING VARIOUS CONDITIONS AND EXCEPTIONS IN THE *INFOBITS* COLUMN OF THE *PROCIMAGES* DATABASE TABLE

Bit	Definition
0	SCAMP failed
1	WCS[a] solution determined to be bad
2	*mShrink* module execution failed
3	*mJPEG* module execution failed
4	No output from *ptfQA* module (as SExtractor found no sources)
5	Seeing was found to be zero; reset it to 2.5″
6	*ptfSeeing* module had insufficient number of input sources
7	No sources found by SExtractor
8	Insufficient number of 2MASS sources in image for WCS verification
9	Insufficient number of 2MASS matches for WCS verification
10	2MASS astrometric R.M.S.E.(s) exceeded threshold
11	SExtractor before SCAMP failed
12	*pv2sip* module failed
13	SCAMP ran normally, but had too few catalog stars
14	SCAMP ran normally, but had too few matches
15	Anomalous low-order WCS terms
16	Track-finder module failed
17	Anomalously high distortion in WCS solution
18	Astrometry.net was run
19	Error from sub runAstrometryDotNet
20	Time limit reached in sub runAstrometryDotNet

[a] World-coordinate system.

The *ptfSEEING* module is then executed in multithreaded mode on different images in parallel. The seeing calculation requires at least 25 sources with the following SExtractor attributes: $FWHM_IMAGE > 0$, a minimum stellarity (*CLASS_STAR*) of 0.8, and *MAG_BEST* flux between 5000 and 50,000 DN. Bit $2^6 = 64$ will be set in the *infobits* column of the *ProcImages* database table (see Table 16) for $ppid = 3$ records associated with science images that contain an insufficient number of sources for the seeing calculation. The *FWHM_IMAGE* values for the vetted sources are histogrammed in 0.1 pixel bins, and the seeing is taken as the mode of the distribution, which is, in practice, the position of the peak bin.

The recomputed seeing is refined relative to the *SEEING* keyword/value that is already present in the header of the camera-image file (see Table 2) and is written to the output FITS header with the keyword *FWHMSEX*, in units of arcseconds. In addition to the selection based on SExtractor parameters described above, the refinements include the benefits of the pixel mask, bias-corrected input data, and proper accounting for saturation.

Lastly, the *ptfMaskCombine* module is executed in multithreaded mode on different masks in parallel, in order to fold the object detections from the SExtractor object check images into the data masks.

The resulting science images are copied to the sandbox and registered in the *ProcImages* database table with pipeline index $ppid = 3$ (see Table 10). The resulting data masks are copied to the sandbox and registered in the *AncilFiles* database table with *anciltype* = "dmask." The science images and their respective data masks are explicitly associated in the latter database table.

9.13. Superflat-Calibration Pipeline

A superflat is a calibration image that corrects for relative pixel-to-pixel responsivity variations across a CCD. This is also known as the nonuniformity correction. Images of different fields observed throughout the night are stacked to build a high signal-to-noise superflat. This process also allows the removal of stars and cosmic rays via outlier rejection and helps average out possible sky and instrumental variations at low spatial frequencies across the input images.

The superflat-calibration pipeline produces a superflat from all suitable science images for a given night, CCD, and filter, after data reduction by the preprocessing pipeline. A minimum of five PTF fields covered by the input images is required to ensure field variegation and effective source removal in the process of superflat generation. Also, a minimum of 10 input images is required, but typically 100–300 images are used to make a superflat. Special logic avoids too many input images from predominantly observed fields in a given night. The resulting superflat is applied to the science images in the image-flattener pipeline (see § 9.14).

The superflat pipeline is wrapped in a Perl script called *superflat.pl*. The database is queried for the relevant preprocessed science images, along with their data masks. The query excludes exposures from the Orion observing program (van Eyken et al. 2011), in which the imaging was of the same sky location for many successive exposures and the telescope dithering was insufficient for making superflats with the data.

The *normimage* module is executed for each preprocessed science image to create an interim image that is normalized by its global median, which is computed after discarding pixel values for which any data-mask bit is set. All normalized values that are less than 0.01 are reset to unity, which minimizes the introduction of artifacts into the superflat.

In order to fit the entire stack of images into available memory (as many as 422 science exposures have been taken in a single night), the *quadrantifyimage* module is executed to break each normalized image into four equally sized subimages. The same module is separately executed for the data masks.

The *createflat* module processes, one quadrant at a time, all of the subimages and their data masks to create associated stack-statistics and calibration-mask subimages. A separate CDF for each CCD provides input parameters for the process (although CCD-dependent processing for superflats is not done at this time, the capability exists). The parameters direct the code, for each pixel location, to compute the median value of the stacked subimage data values (as opposed to some other trimmed average) and the trimmed standard deviation (σ) after eliminating the lower 10% and the upper 10% of the data values for a given pixel (and reinflating the result in accordance with a trimmed Gaussian distribution to account for the data clipping). Lastly, the module recomputes the median after rejecting outliers greater than $\pm 5\sigma$ from the initial median value, as well as computing the corresponding uncertainty. The stack statistics are written to a FITS data cube, where the first plane contains the clipped medians and the second plane contains the uncertainties. The bit definitions for calibration-mask images are given in Table 17.

The *tileimagequadrants* module pieces back together the four quadrants of the stack-statistics and calibration-mask subimages corresponding to each science image. Finally, the *normimage* module is executed on the full-sized stack-statistics image to normalize it by its global image mean and reset any normalized value

TABLE 17

BITS ALLOCATED FOR THE SUPERFLAT CALIBRATION MASK

Bit	Definition
1	One or more outliers rejected
2	One or more NaNs present in the input data
3	One or more data-mask-rejected data values
12	Too many outliers present
13	Too many NaNs present
14	No input data available

NOTE.—Bits not listed are reserved and, for bits 12 and 13, the allowed fraction is currently set to 1.0, so these bits will never be set.

to unity that is less than 0.01 (in the manner described above). The latter module ignores image data that are within 10 pixels of all four image edges in computing the normalization factor.

The pipeline's chief product is a superflat called "superflat .fits." The file is renamed to an archival filename, copied to the sandbox, and registered in the *CalFiles* database table with *caltype* = "superflat." A corresponding ancillary product is also generated: the calibration mask, which is called "superflat_ cmask.fits." The ancillary file is renamed to an archival filename, copied to the sandbox, and registered in the *AncilCal Files* database table with *anciltype* = "cmask."

A number of processing parameters are written to the FITS header of the superflat. These include the number of input images, the outlier-rejection threshold, the superflat normalization factor, and the threshold for unity reset.

Several SDQA ratings are computed for the superflat. These include the following image-data statistics: average, median, standard deviation, skewness, kurtosis, Jarque-Bera test,[19] 15.9 percentile, 84.1 percentile, scale (half the difference between the 84.1 and 15.9 percentiles), number of good pixels, and number of NaN pixels. These values are written to the *SDQA_CalFileRatings* database table. We have found the Jarque-Bera test particularly useful in locating superflats that infrequently contain point-source remnants due to insufficient input data variegation.

9.14. Image-Flattener Pipeline

The image-flattener pipeline's principal function is to apply the nonuniformity or flat-field corrections to the science images. Also, the pipeline runs a process to detect CCD bleeds and radiation hits in the science images (see below), and then executes the *ptfPostProc* module to update the data masks and compute weight images for later source-catalog generation in the frame-processing pipeline (see § 9.15). The pipeline is wrapped in a Perl script called *flattener.pl*. An instance of this pipeline runs on a per-night, per-CCD, per-filter basis. At the beginning of the pipeline, the database is queried for the science images to process, along with their data masks and relevant calibration image, namely, the superflat associated with the night, CCD, and filter of interest. The saturation level for the CCD is also retrieved.

In the rare case that the superflat does not exist, the database function *getCalFiles* searches backward in time, up to 20 nights, for the closest-in-time superflat substitute. In most cases, the superflat made for the previous night is returned for the CCD and filter of interest. Our experience has been that, generally, the superflat changes slowly over time, hence the substitution does not unduly compromise the data.

The *ptfSciencePipeline* module performs the image flattening. It reads in a list of science images and the superflat. It then simply divides each science image by the superflat on a pixel-by-pixel basis. Since the superflat was carefully constructed to contain no values very close to zero, the output image is well behaved, although the processing includes logic to set the image value to NaN in case it has been assigned the representation for infinity. The applied flat is associated with the pipeline products via the *CalFileUsage* database table.

SExtractor is executed to detect CCD bleeds and radiation hits in the science images, and the output check images contain the detections. It is executed on separate science images via seven parallel threads at a time. The saturation level is an important input to this process. The detection method is an artificial-neural-network (ANN) filter. A program called Eye was used to specifically train the ANN on PTF data. Both SExtractor and Eye are freely available.[20]

The *ptfPostProc* module is a pipeline process that, for each science image: (1) updates its data mask and (2) creates a weight image suitable for use in a subsequent SExtractor run for generating a source catalog. The module is executed in multithreaded mode on separate data masks. The superflat, along with the pertinent check image from the aforementioned SExtractor runs, are the other major inputs to this process for a given data mask. The *ptfPostProc* data-mask update includes setting bits to flag CCD bleeds and radiation hits (see Table 15), which are taken to have occurred at pixel locations where check-image values are ≥ 1. Since the check image does not differentiate between the two artifacts at this time, both bits are set in tandem. The *ptfPostProc* weight-map creation starts with the superflat as the initial weight map and then sets the weights to zero if certain bits are set in the data mask at the same pixel location. Pixels in the weight maps that are masked as dead/bad or NaN (see Table 15) consequently will have zero weight values.

Similar to the preprocessing pipeline (see § 9.12), the resulting science images are copied to the sandbox and registered in the *ProcImages* database table with pipeline index $ppid = 10$ (see Table 10), and the resulting data masks are copied to the sandbox and registered in the *AncilFiles* database table with *anciltype* = "dmask." The science images and their respective data masks are explicitly associated in the latter database table. The weight-map files, which are not archived (see § 10.1) but used by the next pipeline (see § 9.15), are copied to the sandbox but not registered in the *AncilFiles* database table.

9.15. Frame-Processing Pipeline

The frame-processing pipeline's major functions are to perform astrometric and photometric calibration of the science images. In addition, aperture-photometry source catalogs are made from the processed science images using SExtractor, and point-spread function (PSF)-fit catalogs are made using DAOPHOT. The processed science images, their data masks, source

[19] The Jarque-Bera test is a goodness-of-fit test of whether a sample skewness and kurtosis are as expected from a normal distribution.

[20] See http://www.astromatic.net for more details.

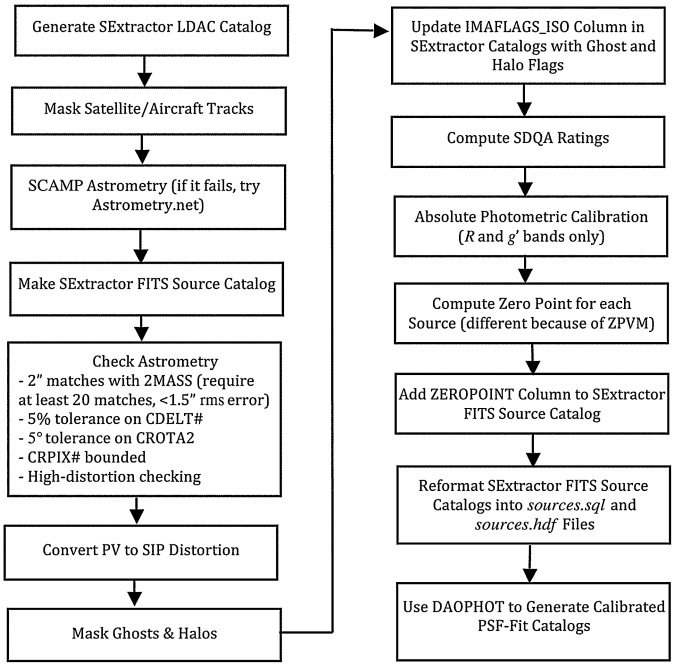

FIG. 7.—Flowchart for the frame-processing pipeline.

catalogs, and other information (such as related to SDQA; see § 8 for more details) are registered in the database to facilitate data analysis and product archiving. Figure 7 shows the flow of data and control through the pipeline.

The frame-processing pipeline is wrapped in a Perl script called *frameproc.pl*. The pipeline begins by querying the database for all flattened science images and associated data masks for the night, CCD, and filter of interest. The files are copied from the sandbox to the pipeline machine's scratch disk for local access. A record for each science image is created in the *ProcImages* database table with pipeline index $ppid = 5$ (see Table 10), which will store important metadata about the processed images, such as a unique processed-image database identification (*pid*), disk location and filename, status, processing version, which version is "best," etc.

The refined seeing computed by the preprocessing pipeline is read from the FITS header (see § 9.12). If its value is zero, then it is reset to 2.5″, and this condition is flagged by setting bit $2^5 = 32$ in the *infobits* column of the corresponding *ProcImages* database record (see Table 16). The refined seeing is a required input parameter for source-catalog generation by SExtractor.

The pipeline next executes SExtractor to generate source catalogs, one per science image, in FITS "LDAC" format (Leiden Data Analysis Center), which is the required format for input to the SCAMP process described below (Bertin 2009). The SExtractor-default convolution filter is applied. The nondefault input configuration parameters are listed in Table 18.

The *createtrackimage* module is executed to detect satellite and aircraft tracks in each science image. Tracks appear with a frequency of a few to several times in a given night and the same

TABLE 18
NONDEFAULT SEXTRACTOR PARAMETERS FOR FITS "LDAC" CATALOG GENERATION

Parameter	Setting
CATALOG_TYPE	FITS_LDAC
DETECT_THRESH	4
ANALYSIS_THRESH	4
GAIN	1.5
DEBLEND_MINCONT	0.01
PHOT_APERTURES	2.0, 3.0, 4.0, 6.0, 10.0
PHOT_PETROPARAMS	2.0, 1.5
PIXEL_SCALE	1.01
BACK_SIZE	32
BACKPHOTO_TYPE	LOCAL
BACKPHOTO_THICK	12
WEIGHT_TYPE	MAP_WEIGHT

track often crosses multiple CCDs. The module looks for contiguous blobs of pixels that are at or above the local image median plus 1.5 times the local image-data dispersion, where the dispersion is computed via the robust method of taking half the difference between the 84.1 percentile and the 15.9 percentile (which reduces to one standard deviation in the case of Gaussian-distributed data). All thresholded pixels that comprise the blobs are tested to ensure they neither are an image-edge pixel nor have their data values equal to NaN or are generally masked out (data-mask bit $2^1 = 2$ for source detections is excepted). The track-detection properties of this module were improved by using local statistics, instead of global, in the image-data thresholding, and our method of computing local statistics, which involves computing statistics on a coarse grid and using bilinear interpolation between the grid points, incurred only a small processing-speed penalty. The *createtrackimage* module utilizes a morphological classification algorithm that relies on pixel-blob size and shape characteristics. The median and dispersion of the blob intensity data are computed, and subsequent morphology testing is done only on pixels with intensities that are within $\pm 3\sigma$ of the median. The blobs must consist of a minimum of 1000 pixels to be track-tested. In order for a blob to be classified as a track, at least one of the following parametrically-tuned tests must be satisfied:

1. The blob length is greater than 900 pixels, or
2. The blob length is $\geq$300 pixels, and the blob half-width is $\leq$10 pixels, or
3. The blog length is greater than 150 pixels, and the blob half-width is less than 2 pixels.

The blob length is found by least-squares fitting a line to the positions of the blob pixels and then computing the maximum extent of the line across the blob. The blob half-width is the robust dispersion of the perpendicular distances between the blob pixels and the fitted line. The data mask associated with the processed image of interest is updated for each track found. The pixels masked as tracks in the data mask are blob pixels that

TABLE 19
COLUMNS IN THE *TRACKS* DATABASE TABLE

Column	Definition
tid	Unique index associated with the track (primary key)
pid	Unique index of the processed image (foreign key)
expid	Unique index of the exposure (foreign key)
ccdid	Unique index of the CCD (foreign key)
fid	Unique filter index (foreign key)
num	Track number in image
pixels	Number of pixels in track
xsize	Track size in x-image dimension (pixels)
ysize	Track size in y-image dimension (pixels)
maxd	Maximum track half-width (pixels)
maxx	Track x-pixel position associated with *maxd*
maxy	Track y-pixel position associated with *maxd*
length	Length of track (pixels)
median	Median of track intensity data (DN)
scale	Dispersion of track intensity data (DN)
a	Zeroth-order linear-fit coefficient of track y vs. x (pixels)
b	First-order linear-fit coefficient of track y vs. x (dimensionless)
siga	Uncertainty of zeroth-order linear-fit coefficient
sigb	Uncertainty of first-order linear-fit coefficient
chi2	χ^2 of linear fit
xstart	Track starting coordinate in x-image dimension (pixels)
ystart	Track starting coordinate in y-image dimension (pixels)
xend	Track ending coordinate in x-image dimension (pixels)
yend	Track ending coordinate in y-image dimension (pixels)

are located within the double-sided envelope defined by four blob half-widths on either side of the track's fitted line. Bit $2^0 = 1$ in the data mask is allocated for flagging track pixels (see Table 15). A record for each track is inserted into the *Tracks* database table; the columns defined for this table are given in Table 19.

The astrometric solution for each science image is computed by SCAMP (Bertin 2009). The star catalog specified as input depends on whether the science image overlaps an SDSS field. The overlap fractions are precomputed and stored in the *FieldCoverage* database table. For the R and g filters, if the fraction equals 1.0, the SDSS-DR7[21] catalog (Abazajian et al. 2009) is selected; otherwise, the UCAC3[22] catalog (Zacharias et al. 2010) is selected. If SCAMP fails to find an astrometric solution, then it is rerun with the USNO-B1[23] catalog (Monet et al. 2003). For the Hα filters, only the UCAC3 catalog is selected. Up to 5 minutes per science image is allowed for SCAMP execution. The process is killed after the time limit is reached, and retry logic allows up to three retries. Since a SCAMP catalog will be the same for a given field, CCD, and filter, the catalogs are cached on disk in a directory tree organized by catalog type and the aforementioned parameters after they are received from the catalog server. The catalog-file cache is therefore checked first before requesting a catalog from the server. Since SCAMP represents distortion using PV coefficients,[24] and some distortion is always expected, the pipeline requires PV coefficients to be present in the FITS-header file that SCAMP outputs as a container for the astrometric solution. The pipeline also parses SCAMP log output for the number of catalog sources loaded and matched and requires more than 20 of these as one of the criteria for an acceptable astrometric solution.

A SCAMP-companion program called MissFITS transfers the astrometric solution to the FITS header of each science-image file. Another process called *hdrupdate* removes the astrometric solution previously found by Astrometry.net from the science-image FITS headers (see § 9.10).

A custom module called *pv2sip* converts the PV distortion coefficients from SCAMP into the Simple Imaging Polynomial (SIP) representation (Shupe et al. 2005). The original code was developed in Python (Shupe et al. 2012) and later translated into the C language by one of the authors (R. R. L.). This pipeline step is needed because WCSTools and other off-the-shelf astronomical software used by the pipeline require SIP distortion coefficients for accurate conversion between image-pixel coordinates and sky coordinates.

The astrometric solution is first sanity-checked and then later verified. The sanity checks, which assure proper constraining of the low-order WCS terms (*CDELT1*, *CDELT2*, *CRPIX1*, *CRPIX2*, and *CROTA2*), are relatively simple tests that are done as described in § 9.10. Regardless of whether the solution is good or bad, the astrometric coefficients are loaded into the *IrsaMeta* database table, which is indexed by processed-image identification (*pid*) and contains the metadata that are required by IRSA (see § 10 below). There is a one-to-one relationship between records in this table and the *ProcImages* database table. Images with solutions that fail the sanity checking will be flagged with $status = 0$ in the *ProcImages* database table, and bit $2^{15} = 32,768$ will be set in the *infobits* column of the *ProcImages* database table (see Table 16). The astrometric verification involves matching the sources extracted from science images with selected sources from the Two Micron All Sky Survey (2MASS) catalog (Skrutskie et al. 2006). A matching radius of 2″ is specified for this purpose. A minimum of 20 2MASS sources must be contained in the image, and the rms error (R.M.S.E.) of the matches, along both image dimensions, must be less than 1.5″. If any of these criteria are not satisfied, then the appropriate bit will be set in the *infobits* column of the *ProcImages* database table (see Table 16), and the image will be flagged as having failed the astrometric verification.

If SCAMP fails to give an acceptable astrometric solution, then Astrometry.net is executed. If this succeeds, then a custom module called *sip2pv* is run to convert the SIP distortion

[21] Sloan Digital Sky Survey, Data Release 7.
[22] The Third U.S. Naval Observatory CCD Astrograph Catalog.
[23] U. S. Naval Observatory B1 Catalog.

[24] The PV distortion coefficients implemented in SCAMP are best documented by Shupe et al. (2012). "PV" is the name assigned by Shupe et al. (2012) for the distortion polynomial that is generated by SCAMP, which creates FITS-header keywords that begin with the suffix "PV".

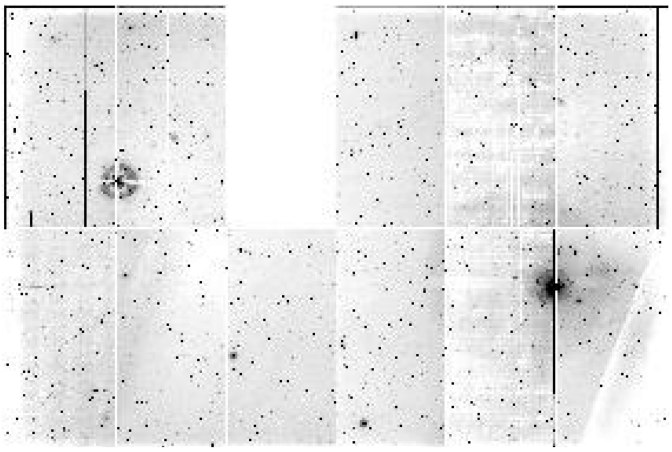

FIG. 8.—Example ghost in PTF exposure $expid = 203381$. The image-display gray-scale table is inverted, so that black indicates high brightness and white indicates low brightness. The large ghost is located in the upper-left portion of the 12-CCD composite image and is imaged onto two CCDs ($ccdid = 4$ and $ccdid = 5$). It is caused by the bright star located in the lower-right portion.

coefficients into PV distortion coefficients, so that the correct source positions are computed by SExtractor when making the source catalogs.

The pipeline includes functionality for inferring the presence of ghosts and halos in R- and g-band images. Ghosts are optical features that are reflections of bright stars about the telescope's optical axis. A bright star imaged in one CCD or slightly outside of the field of view can lead to the creation of a ghost image in an opposite CCD with respect to the telescope boresight. An example ghost is shown in Figure 8. Halos are optical features that surround bright stars and are double reflections that end up offset slightly from the bright star toward the optical axis. An example halo is shown in Figure 9. The ghost positions vary depending on the filter and also whether the image was acquired before or after the aforementioned filter swap (see § 3). Locating these features starts by querying the Tycho-2 catalog and supplement for bright stars, with $V_{\rm mag}$ brighter than 6.2 mag and 9.0 mag for g and R bands, respectively, before the filter swap, and brighter than 7.2 mag for both bands after the filter swap. Ghosts and halos are separately flagged in the data masks associated with processed images. Bit $2^5 = 32$ is reserved for ghosts and bit $2^{12} = 4096$ for halos in the data mask (see Table 15). A circular area is flagged in the data mask to indicate a ghost or halo. Although the ghost and halo sizes vary with bright-star intensity and filter, only a maximally sized circle for a given filter, which was determined empirically for cases before and after the filter swap, is actually masked off. Accordingly, the radius of the circle for a ghost is 170 pixels for the R band (both before and after the filter swap), and, for the g band, is 450 pixels before the filter swap and 380 pixels afterwards. Similarly, the radius of the circle for a g-band halo is 85 pixels before the filter swap and 100 pixels afterwards, and is 95 pixels before and 100 pixels afterwards for R-band halos. Database records in the *Ghosts* and/or *Halos* database tables are inserted for each ghost and/or halo found, respectively.

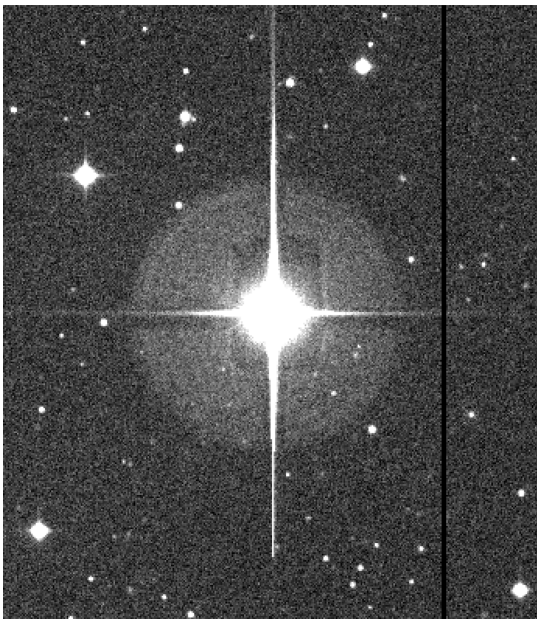

FIG. 9.—Example halo in PTF processed image $pid = 9514402$. Only a portion of the CCD image is shown. The halo surrounding the bright star is $\approx 3'$ in diameter.

Ofek et al. (2012) give a description of the photometric calibration, which is done on a per-night, per-CCD, per-filter basis. The source code for the photometric calibration is written in MATLAB, and the pipeline makes a system call to execute this process. A minimum of 30 astrometrically calibrated science images for the photometric calibration is a software-imposed requirement to ensure adequate solution statistics (sometimes fewer science images are taken in a given night, or an inadequate number could be astrometrically calibrated due to cloudy conditions, etc.). Also, at least 1000 SDSS-matched stars extracted from the PTF-processed images for a given night, CCD, and filter are required for the photometric-calibration process to proceed. The resulting calibration data, consisting of fit coefficients, their uncertainties, and a coarse grid of zero-point-variability-map (ZPVM) values, are loaded into the *AbsPhotCal* and *AbsPhotCalZpvm* database tables and are also written to the pipeline-product image and source-catalog FITS headers. While the source catalogs contain instrumental magnitudes, their FITS headers contain enough information to compute the photometric zero points for the sources, provided that the photometric calibration could be completed successfully. In addition, as elaborated in the next paragraph, we also compute the zero points of individual sources (which vary from source to source because of the ZPVM) and include them in the source catalogs as an additional column; these zero points already include the $2.5\log(\delta t)$ contribution for normalizing the image data by the exposure time, δt, in seconds, and so simply adding the instrumental

magnitudes to their respective zero points will result in calibrated magnitudes. The photometric-calibration process also generates a FITS-file-image version of the ZPVM, which is ultimately archived, and metadata about it is loaded into the *CalFiles* database table with *caltype* = "zpvm." This calibration file is associated with the relevant pipeline products in the *CalFileUsage* database table. The minimum and maximum values in the ZPVM image are loaded into the *AbsPhotCal* database table as additional image-quality measures. There is also a corresponding output FITS file containing an image of ZPVM standard deviations, which is registered in the *CalAncilFiles* database table under *anciltype* = "zpve" and associated with the ZPVM FITS file.

The calculation of the ZPVM contribution to the photometric zero point by the pipeline itself for each catalog source is done via bilinear interpolation of the ZPVM values in the aforementioned grid of coarse cells, which are queried from the *AbsPhotCalZpvm* database table. If any of the values is equal to NaN, which occurs when not enough good matches between PTF-catalog and SDSS-catalog sources are available, then the interpolation result is reset to zero. The ZPVM algorithm requires at least 1000 matches in a 256×256 pixel cell per CCD and filter for the entire night (Ofek et al. 2012), in order to calculate the value for a cell. Because of the ZPVM, the zero point varies from one source to the next. The zero point for each source is written to the SExtractor source catalogs as an additional column, called *ZEROPOINT*.

For each astrometrically calibrated image, SExtractor is executed one last time to generate its final aperture-photometry source catalog. The correct gain and saturation level is set for the CCD of interest. Both detection and analysis thresholds are set to 1.5σ. The input weight map is the superflat with zero weight values where data-mask bits are set for dead, bad, or NaN pixels, as described in § 9.14. The *SEEING_FWHM* option is set to the seeing value computed in 9.14 for each image. A background check image is also generated by SExtractor and stored in the sandbox, in case it is needed as a diagnostic. The nondefault input configuration parameters for SExtractor are listed in Table 20.

Furthermore, for each astrometrically calibrated image, we perform PSF-fit photometry using the DAOPHOT and ALLSTAR software (Stetson 1987). These tools are normally run interactively; however, we have automated the entire process: from source detection to PSF-estimation and PSF-fit photometry in a pipeline script named *runpsffitsci.pl*. Input parameters are the FWHM of the PSF (provided by SExtractor upstream) and an optional photometric zero point. At the time of writing, the input photometric zero point is based on an absolute calibration using the SExtractor catalogs. This is not optimal, and we plan to recalibrate the PSF-fit extractions using calibrations derived from PSF-fit photometry in the near future. The DAOPHOT routines are executed in a single iteration with no subsequent subtraction of PSF-fitted sources to uncover hidden (or missed) sources in a second pass. A spatially varying PSF that is modeled to vary linearly over each image is generated. This is then used to perform PSF-fit photometry. Prior to executing the DAOPHOT routines, the *runpsffitsci.pl* script dynamically adjusts some of the PSF-estimation and PSF-fit parameters, primarily those that have a strong dependence on image quality—the PSF FWHM and image-pixel noise. The default input configuration parameters used for PSF-fit-catalog generation are listed in Table 21. The parameters that are dynamically adjusted are RE, LO, HI, FW, PS, FI, and the A i aperture radii (where $i = 1\ldots6$). In particular, the parameters that depend on the input FWHM (FW) are the linear half-size of the PSF stamp image, PS; the PSF-fitting radius, FI; and the aperture radii Ai, all in units of pixels. These parameters are adjusted according to:

$$PS = \min(19, \mathrm{int}\{\max[9, 6FW/2.355] + 0.5\}),$$
$$FI = \min(7, \max[3, FW]),$$
$$Ai = \min(15, 1.5\max[3, FW]) + i - 1,$$

TABLE 20
NONDEFAULT SEXTRACTOR PARAMETERS FOR FINAL SOURCE-CATALOG GENERATION

Parameter	Setting
CATALOG_TYPE	FITS_1.0
DEBLEND_NTHRESH	4
PHOT_APERTURES	2.0, 4.0, 5.0, 8.0, 10.0
PHOT_AUTOPARAMS	1.5, 2.5
PIXEL_SCALE	1.01
BACKPHOTO_TYPE	LOCAL
BACKPHOTO_THICK	35
WEIGHT_TYPE	MAP_WEIGHT

TABLE 21
DEFAULT INPUT PARAMETERS FOR SCIENCE-IMAGE PSF-FIT-CATALOG GENERATION

daophotsci.opt	photosci.opt
$RE = 15.0$	$A1 = 4.5$
GA $= 1.5$	$A2 = 5.5$
$LO = 10$	$A3 = 6.5$
$HI = 10000.0$	$A4 = 7.5$
$PS = 9$	$A5 = 8.5$
$TH = 2.8$ (30)[a]	$A6 = 9.5$
$VA = 1$	$IS = 2.5$
$EX = 5$	$OS = 20$
$WA = 0$	
$FW = 2.5$	
$FI = 3.0$	
$AN = 1$	
$LS = 0.2$	
$HS = 1.0$	
$LR = -1$	
$HR = 1$	

[a] The TH value in parentheses is for the PSF-creation step.

where $i = 1\ldots6$, "min" and "max" denote the minimum and maximum of the values in parentheses, respectively, and "int" denotes the integer part of the quantity. The *runpsffitsci.pl* script reformats the raw output from DAOPHOT and ALLSTAR and assigns WCS information to each source. The output table is later converted into FITS binary-table format for the archive. The intermediate products, such as the raw PSF file, are written to the sandbox.

After the photometric-calibration process has run and the source catalogs have been created, the pipeline generates a file called *sources.sql*, which contains an aggregation of all SExtractor source catalogs for the night, CCD, and filter of interest. The *sources.sql* file is suitable for use in bulk-loading source-catalog records into the database. However, after extensive testing, it has been determined that loading sources into the PTF operations database is unacceptably slow, and, consequently, this has been temporarily suspended until the PTF-operations network and database hardware can be upgraded. Nevertheless, the file still serves a secondary purpose, which is facilitating the delivery of source information to IRSA, where it is ultimately loaded into an archive relational database. The file contains source information extracted from the final SExtractor source catalogs, as well as a photometric zero point computed separately for each source. In addition, for each source, a level-seven hierarchical-triangular-mesh (HTM) index is computed, and its SExtractor *IMAFLAGS_ISO* and *FLAGS* parameters are packed together, for compact storage, into the upper and lower 2 bytes, respectively, of a 4 byte integer.

A Python process is also run to generate a file with the same data contents as the *sources.sql* file, but in HDF5[25] format. The output from this process is called *sources.hdf*. The HDF5 files can be read more efficiently by Python software and are used in downstream Python pipelines for matching source objects and performing relative photometric calibration.

At the end of this pipeline, the primary products, which are the processed images, are copied to the sandbox and registered in the *ProcImages* database table with the preassigned processed-image database identifications (*pid*) and pipeline index $ppid = 5$ (see Table 10). There is a similar process for ancillary products and catalogs. The ancillary products consist of data masks and JPEG preview images; these are copied to the sandbox and registered in the *AncilFiles* database table with *anciltype* designations of "dmask" and "jpeg," respectively. The catalogs consist of SExtractor and DAOPHOT source catalogs stored as FITS binary tables; these are copied to the sandbox and registered in the *Catalogs* database table with *catType* designations of one and two, respectively. The primary products and their ancillary products and catalogs are explicitly associated with each other by the processed-image database identification, *pid*, in the *AncilFiles* and *Catalogs* database tables. The *sources.sql* and *sources.hdf* files created by the pipeline are copied to the sandbox but not registered in the database. All of these products are included in the subsequent archiving process (see § 10).

[25] http://www.hdfgroup.org/HDF5/whatishdf5.html.

9.16. Catalog-Generation Pipeline

The catalog-generation pipeline is wrapped in a Perl script called *genCatalog.pl* and has been assigned $ppid = 13$ for its pipeline database identification. It performs many, but not all, of the same functions as the frame-processing pipeline (see § 9.15). Most notably, it omits the astrometric and photometric calibrations, because this pipeline expects calibrated input images (which are initially produced by the frame-processing pipeline). The chief purpose of the catalog-generation pipeline is to provide the capability of regenerating source catalogs directly from the calibrated, processed, and archived images and their data masks, for a given night, CCD, and filter. The source catalogs, if necessary, may be produced from different SExtractor and DAOPHOT configurations than were previously employed by the frame-processing pipeline. Also, for the PTF data taken before 2013, only SExtractor catalogs were generated, as the execution of DAOPHOT had not yet been implemented in the frame-processing pipeline. The catalog-generation pipeline is, therefore, intended to also generate the PSF-fit catalogs missing from the archive. Like the frame-processing pipeline, the weight map used by SExtractor in this pipeline to create a source catalog for an input image is generated by starting with a superflat for the weight map and then zeroing out pixels in the weight map that are masked as dead/bad or NaN in the respective data mask of that input image. The pipeline also has functionality for adding and updating information in the FITS headers of the images and data masks. Thus, the products from this pipeline constitute new versions of images, data masks, and source catalogs. The pipeline copies its products to the sandbox and registers them, as appropriate, in the *ProcImages*, *AncilFiles*, and *Catalogs* database tables with pipeline index $ppid = 13$ (see Table 10).

Local copies of the calibration files associated with the input images are made by the pipeline, and these are also copied to the sandbox and associated with the pipeline products in the *CalFiles* and *CalFileUsage* database tables. This ensures that the calibration files are also rearchived when the new products are archived. The reason for this particular approach is technical: the calibration files sit in the directory tree close to the products and are lost when old versions of products are removed from the archive by directory-tree pruning at a high level.

9.17. Reference-Image Pipeline

To help mitigate instrumental signatures and transient phenomena in general at random locations in the individual images (e.g., noisy hardware pixels with highly varying responsivity, cosmic rays, and moving objects, such as asteroids and satellite/aircraft streaks), we co-add the images with outlier rejection to

create cleaner and more "static" representations of the sky. Furthermore, this co-addition improves the overall signal-to-noise ratio relative to that achieved in the individual image exposures.

The reference-image pipeline creates co-adds of input images for the same CCD, filter, and PTF field (*PTFFIELD*). This pipeline is wrapped in Perl script *genRefImage.pl* and is run on an episodic basis as new observations are taken. It has been assigned $ppid = 12$ for its pipeline database identification. Currently, reference images are generated only for the R and g bands.

The candidate input images for the co-adds are selected for the best values of seeing, color term, theoretical limiting magnitude, and ZPVM (see description of absolute photometric calibration in § 9.15). A database-stored function is called to make this selection for a given CCD, filter, and PTF field, and it returns, among other things, the database identifications of candidate processed images that are potentially to be co-added. The input-image selection criteria are listed as follows:

1. All input images must be astrometrically and photometrically calibrated;
2. Exclude inputs with anomalously high-order distortion;
3. Minimum number of inputs = 5;
4. Maximum number of inputs = 50 (those with the faintest theoretical limiting magnitudes are selected);
5. Have color-term values that lie between the first and 99th percentiles;
6. Have ZPVM values between ± 0.15 mag;
7. Have seeing FWHM value $<3.6''$;
8. Have theoretical limiting magnitude >20 mag; and
9. Have at least 300 SExtractor-catalog sources.

The candidate inputs are sorted by limiting magnitude in descending order. An input list is progressively incremented with successive input images, and the resulting co-add limiting magnitude (CLM) is computed after each increment. The objective is to find the smallest set of inputs that comes as closely as possible to the faintest value of CLM from a predefined small set of discrete values between 21.5 and 24.7 mag.

An illumination correction is applied to each selected input image, in order to account for the ZPVM (see § 9.15). Catalogs are generated with SExtractor and then fed to SCAMP all together, in order to find a new astrometric solution that is consistent for all input images.

The co-adder is a Perl script called *mkcoadd.pl*. It makes use of the Perl data language (PDL) for multithreading. The input images and associated data masks are fed to the co-adder. The input images are matched to a common zero point of 27 mag, which is a reasonable value for a 60 s exposure. Thus all PTF reference images have a common zero point of 27 mag. SWarp is used to resample and undistort each input image onto a common fiducial grid based on the astrometric solution (Bertin et al. 2002). Saturated, dead/bad, and blank pixels are rejected. The co-addition procedes via trimmed averaging, weighted by the inverse seeing of each input frame. Ancillary products from the co-adder include an uncertainty image and a depth-of-coverage map.

The astrometric solution is verified against the 2MASS catalog (see § 9.15 for how this is done). The pipeline generates both SExtractor and PSF-fit reference-image catalogs, which are then formatted as FITS binary tables. The PSF-fit catalogs are made using DAOPHOT. Ancillary products from PSF-fit catalog generation include a raw PSF file, a DS9-region[26] file for the PSF-fit sources, and a set of PSF thumbnails arranged on a grid for visualizing the PSF-variation across the reference image. A number of SDQA ratings and useful metadata for IRSA-archiving are computed for the reference image and loaded into the *SDQA_RefImRatings* and *IrsaRefImMeta* database tables, respectively.

At the end of this pipeline, the reference image and associated catalogs and ancillary files are copied to the sandbox. The reference image is registered in the *RefImages* database table with the preassigned reference-image database identification (*rfid*) and pipeline index $ppid = 12$ (see Table 10). The SExtractor and DAOPHOT reference-image catalogs are registered in the *RefImCatalogs* database table with *catType* designations of one and two, respectively. The reference images and their catalogs and ancillary files are explicitly associated with each other by the processed-image database identification, *rfid*, in the *RefImCatalogs* and *RefImAncilFiles* database tables. All of these products are included in the subsequent archiving process (see § 10). The *RefImageImages* database table keeps track of the input images used to generate each reference image.

9.18. Other Pipelines

Other nascent or mature PTF pipelines will be described in later publications. These include pipelines for image differencing, relative photometry, forced photometry, source association, asteroid detection, and large-survey-database loading.

9.19. Performance

As of 2013 August 5, a total of approximately 3.5×10^5 exposures in 1578 nights have been acquired. About 75% of the exposures are on the sky, covering $\approx 2 \times 10^6$ deg^2. There are also fair numbers of bias, dark, and twilight exposures (14.3%, 5.9%, and 4.8%, respectively). Table 22 lists selected pipeline run-time robust statistics broken down by routinely executed pipeline. Recall the $ppid = 7$ pipeline is run on a per-exposure basis, the $ppid = 1$ pipeline is run on a per-night, per-CCD basis, and the remaining pipelines are run on a per-night, per-filter, per-CCD basis, except for the $ppid = 12$ reference-image pipeline, which is run on a per-filter, per-CCD, per-PTF-field basis.

[26] http://ds9.si.edu/site/Home.html.

TABLE 22

SELECTED PIPELINE RUN-TIME STATISTICS (UPDATED ON 2013 AUGUST 5)

ppid[a]	No. of samples	Median (s)	Dispersion[b] (s)
7	339,671	200.4	84.4
1	14,586	85.0	30.2
3	14,840	2201.4	1226.0
4	14,839	1416.5	815.0
10	14,827	4724.1	2424.0
5	14,781	9387.1	6065.0
12	27,890	271.3	70.0

NOTE.—The statistics are pipeline runs on a per-CCD, per-filter, per-night basis, except for the $ppid = 12$ pipeline, which is on a per-CCD, per-filter, per-field basis.

The run-time median and dispersion for all pipelines has changed by less than 10% over the last couple of years or so, with the exceptions of the $ppid = 5$ pipeline, which has become more than 30% slower because of recently added functionality, such as PSF-fit-catalog generation, and the reference-image pipeline, which only came online in the last year.

The performance of our satellite/aircraft track detection algorithm (see § 9.15) has not yet been quantitatively scored in terms of completeness versus. reliability; this will be the subject of a future paper. The algorithm has been tuned to find all tracks at the expense of generating some false tracks. Generally, the false tracks will be associated with long, thin galaxies that mimic tracks or very bright stars having extended CCD bleeds that were not fully masked off in the processing. A large χ^2 of the track's linear fit may indicate a track-proximate bright star with a CCD bleed extending across the track. Multiple records in the *Tracks* database table for the same track in a given image can happen when the data thresholding results in unconnected groups of contiguous pixels along that track.

9.20. Smart-Phone Command and Control

A succinct set of high-level scripted commands was developed to facilitate interrogation and control of the IPAC-PTF software and data system (see Table 23). The commands generate useful short reports and optionally initiate pipeline and archive processes. The low data bandwidth and minimal keyboard typing permitted by these commands makes them ideally suited for execution in a terminal window of a smart phone via cellular data network (a wireless Internet connection is nice, but not required). Of course, the same commands also can be conveniently executed in a personal-computer terminal window.

One of us (R. R. L.), with the help of IPACer Rick Ebert, set up a virtual private network (VPN) on his iPhone to allow secure connections directly to IPAC machines. He also purchased secure-shell program "Prompt, v. 1.1.1" from the Apple Apps Store, which was developed by Panic, Inc. and has since been upgraded, and then installed the app on his iPhone. VPN and "Prompt" are all the software needed to execute the PTF pipeline and archive processes on the iPhone. This set up even enables the execution of low-level commands and arbitrary database queries, albeit with more keyboard typing.

All of the commands listed in Table 23, except for *ptfc*, generate brief reports by default. Some of the commands accept an optional date or list of dates, which is useful for specifying night (s) other than the default current night. Also, some of the commands accept an optional flag, to be set in order for the command to take some action beyond simply producing a report; specifying either no flag or zero for the flag's value will cause the command to take no further action, and specifying a flag value of one will cause the command to perform the action attributed to the command. The *ptfc* command is normally run in the background, by either appending an ampersand character to the command or executing it under the "screen" command.

TABLE 23

HIGH-LEVEL COMMANDS FOR INTERROGATION AND CONTROL OF THE IPAC-PTF SOFTWARE AND DATA SYSTEM

Command	Definition
ptfh	Prints summary of available commands.
ptfi	Checks whether current night has been ingested.
ptfj	Checks status of disks, pipelines, and archiver.
ptfe	Prints list of failed pipelines.
ptfs [YYYY-MM-DD][a][flag (0 or 1)][b]	Launches image-splitting pipelines for given night.
ptff [YYYY-MM-DD] [flag (0 or 1)]	Ignores filter checking and relaunches relevant image-splitting pipelines for given night.
ptfp [YYYY-MM-DD] [flag (0 or 1)]	Launches image-processing pipelines for given night.
ptfr [YYYY-MM-DD] [flag (0 or 1)]	Launches catalog-generation pipelines for given night.
ptfm [YYYY-MM-DD] [flag (0 or 1)]	Launches source-matching pipelines for given night.
ptfq	Prints list of nights ready for archiving.
ptfk [YYYY-MM-DD] [flag (0 or 1)]	Makes archive soft link for given night.
ptfa [list of YYYY-MM-DD]	Schedules processing nights to be archived and generates optional archiver command.
ptfc	Script to manually execute archiver command generated by *ptfa*.
ptfd [YYYY-MM-DD]	Prints delivery/archive information for given night.

[a] The square brackets indicate command options; current date is assumed if no date is specified.

[b] The optional flag set to 1 is required for the command to take action beyond simple report generation.

10. DATA ARCHIVE AND DISTRIBUTION

PTF camera images and processed products are permanently archived (Mi et al. 2013). As was mentioned earlier, the PTF data archive is curated by IRSA. This section describes the processes involved in the ongoing construction of the PTF archive, and, in addition, the user Web interface provided by IRSA for downloading PTF products.

10.1. Product Archiver

The product archiver is software written in Perl, called *productArchiver.pl*, that transfers the latest version of the products from the sandbox to the archive and updates the database with the product archival locations. With the exception of the pipeline log files, all-sky-depth-of-coverage images (Aitoff projections), and nightly aggregated source catalogs (*sources.sql* files), only the processed-image-product files that are registered in the *ProcImages*, *Catalogs*, *AncilFiles*, *CalFiles*, and *CalAncilFiles* database tables are stored permanently in the PTF archive. These include processed images, data masks, source catalogs (FITS binary tables), and JPEG preview images. The calibration files associated with the processed images are also archived. The camera-image files, processed products, and database metadata are delivered to IRSA on a nightly basis. The reference images and associated catalogs and ancillary files are archived with a separate script, with corresponding metadata delivered to IRSA on an episodic basis.

Before the product archiver is executed, a soft link for the night of interest is created to point to the designated archive disk partition. The capacity of the partitions is nominally 8 TB each. The soft links are a convenient means of managing the data stored in the partitions. As new product versions are created and migrated to new partitions, the old partitions, when they are no longer needed, are cleaned out and recycled.

Because both the frame-processing pipeline ($ppid = 5$) and catalog-generation pipeline ($ppid = 13$) produce similar sets of products, but only one set of products for a given night is desirable for archiving, it is necessary to indicate which set to archive. Generally, this is the most recently generated set. The flagging is done by executing a database-stored function called *setBestProductsForNight*, which determines the latest set of products and designates it as the one to be archived. It then sets database column *pBest* in the *ProcImages* database table to one for all best-version records corresponding to the selected pipeline and zero for all best-version records corresponding to the other. Here, one means archive the pipeline products, and zero means do not archive.

The product archiver inserts a record into the *ArchiveVersions* database table, which includes a time stamp for when the archiving started for a particular night, and gets back a unique database identification for the archiving session, named *avid*. The product records for the night of interest in the aforementioned database tables are updated to change *archiveStatus* from 0 to -1, in order to indicate the records are part of a long transaction (i.e., the archiving process for a night's worth of products). After each product has been copied to archival disk storage and its MD5 checksum verified, the associated database record is updated with *avid* and the new file location, and the *archiveStatus* is changed from -1 to 1 to indicate that the product has been successfully archived.

10.2. Metadata Delivery

Database metadata for each night, or for the latest episode of reference-image generation, are queried from the operations database and written to data files for loading into an IRSA relational database. The data files are formatted according to IRSA's specification and then transmitted to IRSA by copying them to a data directory called the "IRSA inbox," which is cross-mounted between PTF and IRSA. The inbox is monitored by a data-ingestion process that is running on an IRSA machine. Separate metadata deliveries are made for camera images, processed images and associated source catalogs, and reference image and associated source catalogs. Source-catalog data for processed images are read from the aggregated *sources.sql* files, rather than queried from the database (since we are not loading source catalogs into the operations database at this time). The creation of the metadata sets is facilitated by database stored functions that marshal the data from various database tables into the *IRSA* database table, which can be conveniently dumped into a data file.

10.3. Archive Executive

The archive executive is software that runs in an open loop on the ingest backup machine. It sequentially launches instances of the VPO (see § 9.6) for each night to be archived. The archive executive expects archive jobs to be inserted as records in the *ArchiveJobs* database table (see § 6). Staging archive jobs for execution, therefore, is effected by inserting associated *ArchiveJobs* database records and assuring that the records are in the required state for acceptance by the executive. The database table is queried for an archive job when the designated archive machine is not currently running an archive job and its archive executive is seeking a new job. The archive job with the latest night date has the highest priority and is executed first. Only one archive job at a time is permitted.

An *ArchiveJobs* database record is prepared for staging an archive job by setting its *status* column to zero. The archive job that is currently executing will have its status set to -1, indicating that it is in a long transaction. The *started* column in the record will also be updated with a time stamp for when the archive job began. Staged archive jobs that have not yet been executed can be manually suspended by setting their status to -1. When the archive job has completed, its status is set to 1, its *ended* column is updated with a time stamp for when the archive

TABLE 24
PRODUCTS IN THE PTF ARCHIVE

Product	Notes
Camera Images	Direct from Mount Palomar; multiextension FITS, per-exposure files.
Processed Images	Astrometrically and photometrically calibrated, per-CCD FITS images.
Data Masks	FITS images with per-pixel bit flags for special data conditions (see Table 15).
Source Catalogs	Both SExtractor and DAOPHOT catalog types in per-CCD FITS binary tables.
Aggregated Catalogs	Nightly aggregated per-CCD SExtractor catalogs, in both SQL and HDF5 formats.
Reference Images	Co-additions of 5+ processed images for each available field, CCD, and filter.
Ref.-Im. Catalogs	Both SExtractor and DAOPHOT catalog types in FITS binary-table format.
Ref.-Im. Ancillary Files	Uncertainty, PSF, and depth-of-coverage maps; DS9-region file for DAOPHOT catalog.
Calibration Files	Superbias, superflat, and ZPVM FITS images for each available night, CCD, and filter.
Sky-Coverage Files	Aitoff FITS images showing per-filter nightly and total observation coverage.
Pipeline Log Files	Useful for monitoring software behavior and tracking down missing products.

job finished, and the *elapsed* column is updated with the elapsed time between starting and ending the archive job.

10.4. Archive Products

At the time of writing, ≈3 million processed CCD images from 1671 nights have been archived. The total number of PTF source observations stored in catalogs is estimated to be more than 40 billion. PTF collaboration members can access the processed products from a Web interface provided by IRSA (see § 10.5).

The archive contains unprocessed camera images, processed images, accompanying data masks, source catalogs extracted from the processed images, reference images, reference-image catalogs, calibration files, and pipeline log files. PTF pipelines generate numerous intermediate product files, but only these final products are stored in the PTF archive. Table 24 provides a complete list of the products that exist in the PTF archive. The archive's holdings include SExtractor and DAOPHOT source catalogs in FITS binary-table files. There are also plans to ingest the catalogs into an IRSA relational database.

10.5. User Web Interface

The PTF-archive Web interface is very similar to the one IRSA provides for other projects,[27] which was in fact built from the same code base. The architecture and key technologies used by modern IRSA Web interfaces have been described by Levine et al. (2009) in the context of the *Spitzer* Heritage Archive.

The PTF archive can be easily searched by sky position, field number, or solar system object/orbit. A batch-mode search function is also available, in which a table of positions must be uploaded. The search results include a list of all PTF data taken over time that match the search criteria. Metadata about the search results, such as when the observations were made, is returned in a multicolumn table in the Web browser. The table currently has more than a dozen different columns. The search results can be filtered in specific ranges of the metadata using the available Web-interface tools.

The Web interface has extensive FITS-image viewing capabilities. When a row in the metadata table is selected, the corresponding processed image is displayed.

The desired data can be selected using check boxes. There is also a check box to select all data in the search results. The selected data are packaged in the background, and data downloading normally commences automatically. As an option, the user can elect instead to be e-mailed the URL for downloading at some later convenient time.

11. LESSONS LEARNED

The development and operations of the IPAC-PTF image processing and data archiving has required one to two software engineers to design custom source code, a part-time pipeline operator to utilize the software to generate and archive the data products on a daily basis, a part-time hardware engineer to set up the machines and manage the storage disks, a part-time database administrator to provide database consulting and backup services, and four to six scientists to recommend processing approaches and analyze the data products. The team breakdown in terms of career experience is roughly 70% seasoned senior and 30% promising junior engineers and scientists. The small team allows extreme agility in exploring data-processing options and setting up new processes. Weekly meetings and information sharing via a variety of database-centric systems (e.g., wiki, operations-database replicate, software-change tracking) have been key managerial tools of a smoothly running project. Teleconferences are not nearly as effective as face-to-face meetings for projects of this kind. Software documentation has been kept minimal to avoid taxing scarce resources. Separate channels for providing products to "power users" closer to the center of the organization versus regular consumers of the products have enhanced productivity and improved product quality on a faster timescale. The necessity of having engineers actually run the

[27] For example, see http://irsa.ipac.caltech.edu/applications/wise.

software they write on a daily basis has significantly narrowed the gap between engineering and operational cultures within the team. While discipline is needed in making good use of the software version-control and change-tracking systems, and in releasing upgraded software to operations, a CCB (change-control board) has not been needed thus far. This kind of organization may not work well in all settings, but it has worked very well for us. Also, as data flow seven days a week, it is good to have someone on the team who is willing to work outside normal business hours, such as doing urgent weekend builds and monitoring the image processing.

The PTF system is complex, and weeding out problems with a small team and very limited resources has been a challenge. To the extent possible, we have followed best practices with an astronomy perspective (Shopbell 2008). Several specific lessons learned are described in the following paragraphs.

Inspecting the data for issues could absorb a tremendous amount of time; still, this time is very well spent, and it is important to make the process as efficient as possible to maximize the benefits from this inspection. A balanced approach that examines the data products more or less evenly, with perhaps slightly more emphasis on the higher-level data products has been a good strategy. Analyzing the products and writing science papers for professional journal publication is probably the best way to bring data issues to light; in fact, this method has unearthed subtle flaws in the processed products that would have otherwise gone unnoticed and suggests that a narrow partnership between those writing science papers and those developing the software is an essential ingredient for success in any data-processing project.

We found it advantageous to wrap all pipeline-software database queries in stored functions and put them all in a single source-code file. This makes it a much less daunting task to later review the database functionality and figure out the necessary optimizations. The single source-code file also facilitates viewing the database functionality as a coherent unit at a point in time. Past versions of this file, which obviously have evolved over time, can be easily checked out from the CVS repository.

Pipeline configuration and execution must be kept simple, in order for those who are not computer scientists to be able to run pipelines themselves outside of the pipeline-executive apparatus. Having several sandbox disks available for storing pipeline products is invaluable because the pipelines can be run on many cases to test various aspects of the pipelines and the data. Equipping pipeline users with a means of configuring the database and sandbox disk for each pipeline instance allows greater flexibility.

Isolating products on disk and in the database according to their processing version is very important, a lesson learned from the *Spitzer* project. Our database schema and stored functions are set up to automatically create product records with new version numbers, and these version numbers are incorporated into disk subdirectory names for uniqueness. Occasionally, a pipeline for a given CCD will fail for various reasons, and it is necessary to rerun the pipeline just for that CCD. This is possible with our pipeline and database design. Having multiple product versions in the sandbox can be extremely useful, provided they are clearly identified, in separate, but nearby, data directories, and database queryable. This, of course, requires the capability of querying the database for the best-version products before pulling the trigger to archive a night's worth of products. It is also very useful to be able to locate the products in a directory tree without having to query a database for the location.

The little details of incorporating the right data in the right places really do matter. Writing more diagnostics rather than less to a pipeline log file provides information for easier software debugging. The diagnostics should include time stamps and elapsed times to run the various processes, as well as CDF listings and module command-line arguments. The aforementioned product versioning is crucial to the data management, and so is having the software and CDF version numbers written to both the product's database record and its FITS header, which aids not only debugging, but also data analysis. It is not fully appreciated how useful these things are unless one actually performs these tasks.

Being able to communicate with the image-processing and archiving system remotely results in great cost savings because it lessens the need to have reserve personnel to take over when the pipeline operator is away from the office. Ideally, the software that interfaces to the system will be able to deliver reports and execute commands with a low-bandwidth connection. Text-based interfaces rather than GUIs simply function better under a wider range of conditions and situations. Our setup includes these features, and even works for cases where direct Internet is unavailable, but cellular communications allow access (see § 9.20). We have demonstrated its effectiveness when used from the home office and from remote locations, such as observatory mountaintops.

Another lesson learned is that problems occur no matter how fault tolerant the system (e.g., power outages). Rainy-day scenarios must be developed that prescribe specific courses of action for manual intervention when automated processing is interrupted. Sometimes the cause of a problem is never found, in which case work-arounds to deal with the effects must be implemented as part of the automated system (e.g., rerunning pipelines that randomly fail with a "signal 13" error). Sometimes the problem goes away mysteriously, obviating the need for a fix or work-around. Other problems have known causes, but cannot be dealt with owing to lack of resources; e.g., an inexpensive router that drops packets or network limitations of the institutional infrastructure. The latter example led to periodically slow and unpredictable network data-transfer rates, which is one of the reasons we stopped loading source-catalog records into the operations database.

Here is a summary of takeaway lessons and recommendations for similar large telescope projects:

1. Pipeline software development is an ongoing process that continues for years beyond telescope first light.
2. A development team in frequent face-to-face contact is highly recommended.
3. The engineering and operations teams should work closely together and be incentivized to "take ownership" of the system.
4. A closely-coupled relational database is essential for complex processing and data management.
5. Pay special attention to how asynchronous camera-exposure metadata are combined with camera images, in order to assure that the correct metadata is assigned to each image.
6. Low-bandwidth control of pipeline job execution is useful from locations remote to the data center.
7. Be prepared to work around problems of unknown cause.
8. There will be a robust demand from astronomers for both aperture-photometry and PSF-fit calibrated source catalogs, as well as reference images and associated catalogs, light-curve products, and forced-photometry products.
9. Scientists studying the data products are an effective science-driven means of finding problems with the data and processing.
10. The data network is a potential bottleneck and should be engineered very carefully, both from the mountain and within the data center.

12. CONCLUSIONS

This paper presents considerable detail on PTF image processing, source-catalog generation, and data archiving at IPAC. The system is fully automated and requires minimal human support in operations, since much of the work is done by software called the "virtual pipeline operator." This project has been a tremendous success in terms of the number of published science papers (80 and counting). There are almost 1500 field and filter combinations (mostly R band) in which more than 50 exposures have been taken, which typically occurred twice per night. This has allowed unprecedented studies of transient phenomena from asteroids to supernovae. More than three million processed CCD images from 1671 nights have been archived at IRSA, along with extracted source catalogs, and we have leveraged IRSA's existing software to provide a powerful Web interface for the PTF collaboration to retrieve the products. Our archived set of reference (co-added) images and catalogs numbers over 40 thousand field/CCD/filter combinations and is growing as more images that meet the selection criteria are acquired. We believe the many design features of our PTF-data processing and archival system can be used to support future complex time-domain surveys and projects. The system design is still evolving, and periodic upgrades are improving its overall performance.

E. O. O. is incumbent of the Arye Dissentshik career development chair and is gratefully supported by grants from the Israeli Ministry of Science, the Israeli Centers of Research Excellence (I-CORE) Program of the Planning and Budgeting Committee, and the Israel Science Foundation (grant No. 1829/12). We wish to thank Dave Shupe, Trey Roby, Loi Ly, Winston Yang, Rick Ebert, Rich Hoban, Hector Wong, and Jack Lampley for valuable contributions to the project. PTF is a scientific collaboration between the California Institute of Technology, Columbia University, Las Cumbres Observatory, the Lawrence Berkeley National Laboratory, the National Energy Research Scientific Computing Center, the University of Oxford, and the Weizmann Institute of Science. This work made use of Montage, funded by the NASA's Earth Science Technology Office, Computation Technologies Project, under Cooperative Agreement Number NCC5-626 between NASA and the California Institute of Technology. Montage is maintained by the NASA/IPAC Infrared Science Archive. This project makes use of data from the Sloan Digital Sky Survey, managed by the Astrophysical Research Consortium for the Participating Institutions and funded by the Alfred P. Sloan Foundation, the Participating Institutions, the National Science Foundation, the US Department of Energy, NASA, the Japanese Monbukagakusho, the Max Planck Society, and the Higher Education Council for England. This research has made use of the VizieR catalog access tool, Centre de Données (CDS), Strasbourg, France. Our pipelines use many free software packages from other institutions and past projects (see Table 12), for which we are indebted.

APPENDIX.

SIMPLE PHOTOMETRIC CALIBRATION

PTF pipeline processing executes two different methods of absolute photometric calibration. We implemented a simple method early in the development, which is documented below. It is relevant because its results are still being written to the FITS headers of PTF processed images. Later, we implemented a more sophisticated method of photometric calibration, which is described in detail by Ofek et al. (2012) and whose results are also included in the FITS headers. For both methods, the SDSS-DR7 astronomical-source catalog (Abazajian et al. 2009) is used as the calibration standard. The simple method is implemented for the R and g camera filters only, and there are no plans to extend it to other filters. The zero point derived from the former method, which is executed for each CCD and filter on the associated data taken in a given night, provides a useful sanity check on the same from the latter method, which are complicated by small variations in the zero point from one image position to another.

A1. DATA MODEL AND METHOD

Our simple method is a multistep process that finds a robust photometric calibration for astronomical sources from fields overlapping SDSS fields. For a given image, we assume there are N source data points indexed $i = 0, \ldots, N-1$ and, for each data point i, the calibrated SDSS magnitude $M_i^{\rm SDSS}$ and the PTF instrumental (uncalibrated) magnitude $M_i^{\rm PTF}$ for the same filter are known. We also make use of the color difference $g_i - R_i$ from the SDSS catalog. The data model is

$$M_i^{\rm SDSS} - M_i^{\rm PTF} = ZP + b(g_i - R_i). \tag{A1}$$

The model parameters are the photometric-calibration zero point ZP and the color-term coefficient b. The latter term on the right-hand side of equation (A1) represents the magnitude difference due to the difference in spectral response between like PTF and SDSS filters.

Radiation hits, optical ghosts and halos, and other data artifacts can have an adverse effect on the data-fitting results of conventional least-squared-error minimization. To introduce a robust measure, a Lorentzian probability distribution function is assumed for the error distribution of the matched astronomical sources:

$$f \propto \frac{1}{1 + (1/2)z^2}, \tag{A2}$$

where

$$z = \frac{y_i - y(g_i - R_i|ZP, b)}{\sigma_i}. \tag{A3}$$

In the numerator of equation (A3), y_i represents the left-hand side of equation (A1), while $y(g_i - R_i|ZP, b)$ represents the right-hand side of the same. In its denominator, σ_i is the standard deviation of y_i.

Using straightforward maximum-likelihood-estimation analysis, the cost function to be minimized by varying ZP and b reduces to

$$\Lambda = \sum_{i=0}^{N-1} \log\left(1 + \frac{1}{2}z^2\right). \tag{A4}$$

Equation (A4) has the advantage of decreasing the weight for outliers in the tails of the data distribution, whereas the Gaussian-based approach will give more weight to these points, thus skewing the result.

A2. IMPLEMENTATION DETAILS

Astronomical sources are extracted from PTF processed images using SExtractor. We elected to use a fixed aperture of 8 pixels (8.08″) in diameter in the aperture-photometry calculations that yield the PTF instrumental magnitudes, which are derived from SExtractor*'s FLUX_APER* values. The PTF sources used in the simple photometric calibration are selected on criteria involving the following SExtractor parameters: $FLAGS = 0$, $CLASS_STAR \geq 0.85$, and *FLUX_MAX* is greater than or equal to 4 times *FLUX_THRESHOLD*. The selected PTF sources, therefore, are unflagged, high signal-to-noise stars. These stars are matched to sources in the SDSS-DR7 catalog with a matching radius of 2″, and a minimum of 10 matches are required, in order to execute the simple

TABLE 25
FITS KEYWORDS ASSOCIATED WITH OUR SIMPLE PHOTOMETRIC CALIBRATION

FITS keyword	Definition
PHTCALEX	Flag set to 1 if simple photometric calibration was executed without error. The flag is set to zero if either there was an execution error or it was not executed.
PHTCALFL	Flag for whether the image is from what was deemed a "photometric night," where 0 = no and 1 = yes (see subsection A2 for more details).
PCALRMSE	Rms error from data fitting with equation (A5), in physical units of magnitude.
IMAGEZPT	Image zero point, in physical units of magnitude, either computed with equation (A5) or taken directly from the data fitting with equation (A1), depending on whether the image overlaps an SDSS field. The keyword's value is set to NaN if $PHTCALEX = 0$.
COLORTRM	Color-term coefficient b, in dimensionless physical units, from equation (A1). This keyword will not be present in the FITS header unless the image overlaps an SDSS field.
ZPTSIGMA	Robust dispersion of $M_i^{\rm SDSS} - M_i^{\rm PTF}$ after data fitting with equation (A1), in physical units of magnitude. This keyword will not be present in the FITS header unless the image overlaps an SDSS field.
IZPORIG	String set to "SDSS" if the image overlaps an SDSS field and *IMAGEZPT* is from equation (A1) or set to "CALTRANS" if the image does not overlap an SDSS field and *IMAGEZPT* is from equation (A5) or set to "NotApplicable" if $PHTCALEX = 0$.
ZPRULE	String set to "DIRECT" if the image overlaps an SDSS field and *IMAGEZPT* is from equation (A1) or set to "COMPUTE" if the image does not overlap an SDSS field and *IMAGEZPT* is from equation (A5) or set to "NotApplicable" if $PHTCALEX = 0$.
MAGZPT	Zero point at an air mass of zero, in physical units of magnitude. Set to NaN if $PHTCALEX = 0$. Note that the keyword's comment may state it is the zero point at an air mass of 1, which is regrettably incorrect.
EXTINCT	Extinction coefficient, in physical units of magnitude. Set to NaN if $PHTCALEX = 0$.

photometric calibration. The flux densities of the stars and associated uncertainties are normalized by their image exposure times.

Two steps are taken to perform the data fitting based on the data model described in subsection A1. First, a simple linear regression with Gaussian errors is performed as an initial input for the robust regression. The Lorentzian error regression analysis is then performed using a Nelder-Mead downhill simplex algorithm with these initial values for the zero point and color-term coefficient. This algorithm has proven to be quite robust, with a 5%–10 % failure rate when the precision is set to the machine epsilon. This rate drops to nearly zero when the precision is set to a factor of 10 times the machine epsilon.

Only for images overlapping SDSS fields is the method of subsection A1 performed. Regardless of SDSS-field overlap, the images will each have a unique air mass value A. The photometric-calibration results are thus treated as a function of air mass, and by employing a linear data model, a zero point at an air mass of zero and an air-mass extinction coefficient are then computed nightly for each CCD and filter (data acquisition for both g and R filters in the same night is possible). These quantities are obtained by a similar linear-regression method, where the data fitting is done with the following first-order polynomial function of air mass $ZP(A)$, where the zero point at an air mass of zero is the zeroth-order fit coefficient $ZP^{A=0}$ and the extinction coefficient is the first-order coefficient β:

$$ZP(A) = ZP^{A=0} - \beta A. \tag{A5}$$

This equation is used to obtain the zero point for images that do not overlap SDSS fields. For the images that do, the zero point from subsection A1 is used directly. The data model is formulated so that the extinction coefficient will normally be a value greater than zero.

The software also makes a determination on whether the night is "photometric" for a given CCD and filter. The basic ad hoc criterion for this specification is that the extinction coefficient must be a value in the 0.0–0.5 range. Additionally, we require a Pearson's r-correlation above 0.75.

To apply the zero point for converting from SExtractor instrumental magnitude to calibrated magnitude, the following equation is used:

$$M_{\rm Cal}^{\rm PTF} = M_{\rm SEx}^{\rm PTF} + ZP + 2.5\log_{10}(T_{\rm exp}), \tag{A6}$$

where $T_{\rm exposure}$ is the exposure time of the associated image, in seconds. If the color difference $g_i - R_i$ for a source is known, then the color term can also be included in the application of the simple photometric calibration; otherwise, it is ignored.

TABLE 26

STATISTICS OF THE RESULTING COLOR-TERM COEFFICIENTS COMPUTED FROM THE SIMPLE PHOTOMETRIC CALIBRATION (SEE EQ. [A1]), BROKEN DOWN BY CCD AND FILTER

CCDID	Filter	N (counts)	Average (dimensionless)	Std. Dev. (dimensionless)
0	g	23,172	0.1786	0.0962
	R	125,604	0.1457	0.0817
1	g	23,247	0.1134	0.1002
	R	126,034	0.1482	0.0758
2	g	23,265	0.1290	0.0919
	R	125,991	0.1416	0.0692
4	g	23,066	0.1158	0.0904
	R	125,205	0.1335	0.1069
5	g	23,140	0.1812	0.0852
	R	125,376	0.1283	0.1311
6	g	23,044	0.1103	0.0925
	R	125,453	0.1500	0.0613
7	g	23,073	0.1027	0.1089
	R	125,613	0.1424	0.0775
8	g	23,092	0.1018	0.0986
	R	126,013	0.1345	0.0795
9	g	23,243	0.1129	0.0958
	R	125,318	0.1097	0.1466
10	g	23,052	0.0993	0.0933
	R	124,806	0.1406	0.0913
11	g	22,775	0.1775	0.0927
	R	124,275	0.1415	0.0743

Table 25 lists the FITS keywords associated with our simple photometric calibration, which are written to the headers of the image files.

A3. PERFORMANCE

The simple method yields a photometric calibration of reasonable accuracy. Of the R-band nights that could be calibrated, where typically more than 50 CCD images that overlap SDSS fields were acquired, half of the nights had a zero-point standard deviation of less than 0.044 mag across all magnitudes and CCDs, and 70% of them had a standard deviation of less than 0.105 mag. The mode of the distribution of nightly zero-point standard deviations is 0.034 mag. On the other hand, 22% of the nights had a standard deviation >1 mag. This range is larger than the 0.02–0.04 mag accuracy reported by Ofek et al. (2012) for our more sophisticated method. Yet, under favorable conditions, simple photometric calibration works remarkably well.

From a sample of approximately 1.66 million data points, we can evaluate the statistics of the free parameters in equation (A5). The average $ZP^{A=0}$ is 23.320 mag, with a standard deviation of 0.3144 mag. The average β is 0.1650 mag per unit air mass, with a standard deviation of 0.3019 mag.

The coefficient b has been found empirically to fall into a relatively small range of values. Table 26 gives statistics of the color-term coefficient broken down by CCD and filter.

REFERENCES

Abazajian, K. N., et al. 2009, ApJS, 182, 543

Arcavi, I., et al. 2010, ApJ, 721, 777

Bertin, E. 2006a, SExtractor User's Manual, Version 2.5, (Institut d'Astrophysique & Observatoire de Paris)

———. 2006b, in ASP Conf. Ser. 351, Astronomical Data Analysis and Software Systems (ADASS) XV, ed. C. Gabriel, et al. (San Francisco: ASP), 112

———. 2009, SCAMP User's Guide, Version 1.6, (Institut d'Astrophysique de Paris)

Bertin, E., & Arnouts, S. 1996, A&AS, 117, 393

Bertin, E., Mellier, Y., Radovich, M., Missonnier, G., Didelon, P., & Morin, B. 2002, in ASP Conf. Ser. 281, Astronomical Data Analysis and Software Systems XI, ed. D. A. Bohlender, D. Durand, & T. H. Handley (San Francisco: ASP), 228

Grillmair, C. J., et al. 2010, in ASP Conf. Ser. 434, Astronomical Data Analysis and Software Systems XIX, ed. Y. Mizumoto (San Francisco: ASP), 28

Holwerda, B. W. 2005, Source Extractor for Dummies (5th ed; Baltimore: STSCi)

Laher, R. R., Levine, D., Mannings, V., McGehee, P., Rho, J., Shaw, R. A., & Kantor, J. 2009, in ASP Conf. Ser. 411, Astronomical Data Analysis and Software Systems (ADASS) XVIII, ed. D. Bohlender, D. Durand, & P. Dowler (San Francisco: ASP), 106

Lang, D., Hogg, D. W., Mierle, K., Blanton, M., & Roweis, S. 2010, AJ, 139, 1782

Law, N. M., et al. 2009, PASP, 121, 1395

———. 2010, Proc. SPIE 7735, 77353 M

Levine, D., et al. 2009, in ASP Conf. Ser. 411, Astronomical Data Analysis and Software Systems (ADASS) XVIII, ed. D. Bohlender, D. Durand, & P. Dowler (San Francisco: ASP), 29

Mi, W., et al. 2013, in Databases in Networked Information Systems, ed. A. Madaan, S. Kikuchi, & S. Bhalla (Berlin Heidelberg: Springer), 67

Monet, D. G., et al. 2003, AJ, 125, 984

Nugent, P. E., et al. 2011, Nature, 480, 344

Ofek, E. O., et al. 2012, PASP, 124, 62

Rahmer, G., Smith, R., Velur, V., Hale, D., Law, N., Bui, K., Petrie, H., & Dekany, R. 2008, Proc. SPIE, 7014, 70144 Y

Rau, A., et al. 2009, PASP, 121, 1334

Sesar, B., et al. 2012, ApJ, 755, 134

Shopbell, P. L. 2008, in ASP Conf. Ser. 394, Astronomical Data Analysis and Software Systems (ADASS) XVII, ed. R. W. Argyle, P. S. Bunclark, & J. R. Lewis (San Francisco: ASP), 738

Shupe, D. L., Laher, R. R., Storrie-Lombardi, L., Surace, J., Grillmair, C., Levitan, D., & Sesar, B. 2012, Proc. SPIE, 8451, 84511 M

Shupe, D. L., Moshir, M., Makovoz, D., & Narron, R. 2005, in ASP Conf. Ser. 347, Astronomical Data Analysis and Software Systems (ADASS) XIV, ed. P. L. Shopbell, M. C. Britton, & R. Ebert (San Francisco: ASP), 491

Skrutskie, M. F., et al. 2006, AJ, 131, 1163

Stetson, P. B. 1987, PASP, 99, 191

Tody, D. 1986, Proc. SPIE, 627, 733

———. 1993, in ASP Conf. Ser. 52, Astronomical Data Analysis and Software Systems II, ed. R. J. Hanisch, R. J. V. Brissenden, & J. Barnes (San Francisco: ASP), 173

van Eyken, et al. 2011, AJ, 142, 60

York, D. G., et al. 2000, AJ, 120, 1579

Zacharias, N., et al. 2010, AJ, 139, 2184