Pan-STARRS Document Control
PSDC-430-???-DR

UNIVERSITY OF HAWAII AT MĀNOA
Institute for Astrononmy

Pan-STARRS Project Management System

Pan-STARRS Image Processing Pipeline

Eugene A. Magnier, Paul A. Price, Joshua Hoblitt
Pan-STARRS Image Processing Pipeline
Institute for Astronomy
June 19, 2007

Revision History

Revision Number	Release Date	Description
DR	2007-01-31	First draft

TBD Listing

Section No.	Page No.	TBD No.	Description
2.4.1	7	1	How to specify more than one fringe mode?
2.4.1	7	2	Discuss blocks, re-queuing science images, analysis versions
2.4.1	7	3	Illustrate how to specify the DVO output database
2.4.2	7	4	Define an inject mechanism that uses Nebulous
3	8	5	Need to document command-line arguments, perhaps even algorithms.
5.1.1.4.2	15	6	set this up for better discovery

Contents

List of Figures

1 Overview

The Pan-STARRS Image Processing Pipeline (IPP) is a software suite for the reduction of astronomical images. Although designed principally for the Pan-STARRS project, it is highly configurable and extensible, and hence highly applicable for other projects.

The IPP consists of a number of elements: programs which perform specific image analysis steps; a parallel processing environment in which the image processes steps are automatically sequenced and tracked; a databasing system for associating and calibrating the detections of astronomical objects. These programs use a number of internal and external libraries (see §6.2), simplifying the process of adding additional elements as needed.

The core functionality of the IPP is implemented by NN principal programs: `psphot` is used for the detection and analysis of objects in astronomical images; `psastro` performs the astrometric calibration of the images; `ppImage` is used for a wide range of single image analysis step, include image detrending and the generation of Q/A images and plots; `ppMerge` combines collections of input detrend images into high-quality masters; `pswarp` transforms images between different pixel projections and coordinate systems; `ppStack` is used to combine multiple science images, including outlier rejection which is sensitive to varying image quality; `ppSub` performs image differencing incorporating the variations in seeing with multiple optional convolution kernels.

In addition to the image-level analysis, the IPP provides DVO, the Desktop Virtual Observatory, a database system for tracking astronomical objects and detections. DVO includes tools for querying and manipulating the contents of the object database. It also provides a number of tools for performing the analysis of quantities at the database level. These include: `relphot`, which performs relative photometry and calculates robust ensemble photometry for objects; `relastro`, which calculates average astrometry quatities for objects, including proper motion and parallax, as well as iterative improvements to the astrometric calibrations of images injested by the database; `uniphot`, which is used to calculate consistent photometric calibrations and transformations.

The IPP provides a system for bulk automation of all stages of the image analysis process within a parallel processing environment. The parallelization scheme is very light-weight, and makes use of distributed UNIX jobs operating on multiple machines within a cluster. Process scheduling and distribution of the resulting jobs to the parallel cluster is performed by `pantasks`. A set of `pantasks` scripts, `ippTasks`, is used to define the processing stages.

Data flow within the IPP is managed via interaction with a collection of database tables, representing the steps of the analysis pipeline. The IPP examines the state of these database tables to determine which jobs should be performed next. A set of simple text files (`dbconfig`) defines the database scheme and are also used to automatically generate C code used to query the database tables. Higher level command-line programs (`ippTools`), built on these APIs, are available to both the end user and to `pantasks`) to examine the state of the pipeline database. A set of Perl scripts (`ippScripts`) provide the glue between the individual IPP analysis programs and the parallel processing environment. A web-based tool, `ippMonitor` provides the user interface for monitoring the pipeline and the current status of the data analysis process.

this is not needed here: move elsewhere

`psLib` is the Pan-STARRS library, containing a range of low-level structures (e.g., vectors, images, etc.) and functions (e.g., write an image to FITS file), most of which are not specific to astronomy. `psModules` is built on top of psLib, and provides higher-level capabilities that are centered around astronomical data manipulation.

We have some Perl modules which are used to facilitate the data flow in processing: `PS::IPP::Metadata::Config` reads "metadata configuration" files; `PS::IPP::Metadata::Stats` interprets statistics from `ppStats`; `PS::IPP::Metada` interprets output lists from the `ippTools`; and `PS::IPP::Config` reads the configuration files.

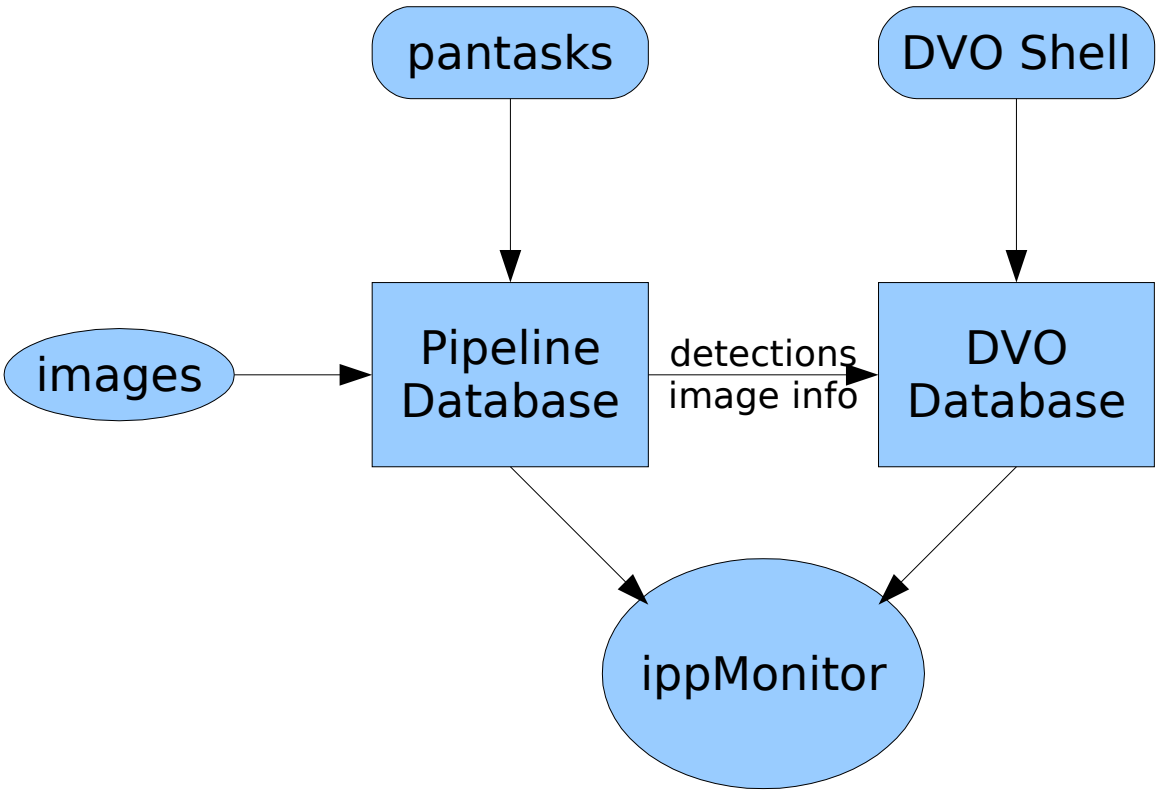

Figure 1: User's view of the IPP

2 Using the IPP

2.1 Conceptual Overview

The operating goals of the IPP can be broken down into several major categories:

- reduction, analysis, and calibration of individual astronomical science images.
- combinations (additions and subtractions) of groups of science image, along with their analysis and calibration.
- analysis of the ensemble properties of astronomical objects detected from many images, including improved astrometric and photometric calibrations.
- construction of the necessary master detrend images and other calibration information needed to perform the science analysis.
- enabling further investigation into the data characteristics by tracking sufficient metadata and providing tools for this analysis.

Furthermore, the IPP is designed to perform these tasks with a high degree of automation and for large collections of data from multiple instruments.

A cartoon of the user's view on the IPP is seen in Figure 1. Images are injected into the IPP, either automatically by software interacting with the telescope systems or manually by the user. The images are processed by the IPP through

any number of analysis steps; the specifics of the analysis will depend on the type of image, the overall configuration, and additional requests the user may make. The user may use the program `pantasks` to control and monitor the ongoing processing, or view summary result information in the ippMonitor web browser. As science images are processed, the information about the objects detected in these images is passed to the DVO database. The user may explore the results or perform further science analysis by interacting with the DVO database via the DVO Shell. From this tool, the user may, for example, create color-magnitude diagrams of regions in the sky, or plot light curves of specific objects. The DVO Shell makes it possible to performing data extraction and data visualization on the DVO database.

Below, we discuss in more detail how a user would use the IPP in several different operating scenarios: to reduce the results from an individual observing run; to perform automatic analysis of data from a telescope with a blocked-out schedule, as for example the CFHT Megacam Queue; to perform automatic analysis of data from a continuous survey instrument such as the Pan-STARRS PS1 Telescope. These difference scenarios have slightly different conditions and underlying assumptions, and make a good set of examples for the types of decisions a user much make.

All of these scenarios have a few common elements. The first goal of the user is to explore the characteristics of the instrument in question and to define initial master detrend images. The intial master detrend images may be used to test the validity of new detrend images and thereby test the evolution of the instrument response. Next, the user will make an initial analysis of science images. This initial analysis may be used to construct further master detrend or calibration information (eg, flat-field correction images; reference astrometric parameters; deeper astrometric reference catalog). Finally, the user may begin large-scale automatic analysis of the complete data collection or the real-time data stream.

2.2 Projects and Databases

The IPP has the concept of a 'project': a related group of data that are analysed within the same context. A project may consist of data from many different camera, or the user may find it more convenient to divide data even from the same camera into multiple independent projects. A single installation of the IPP may operate on many separate projects at the same time.

Within a single project, the user may also choose to define more than one output DVO database. For example, within the Pan-STARRS project, the detections from the different survey modes, with different depths and cadences, will initially be saved in separate DVO databases, and only later joined for improved calibration and more in-depth analysis.

2.3 User Interface Tools

In this section, we introduce the user to the IPP user interface tools. These are the elements of the IPP which provide feedback to the user or allow the user to initiate a new type of analysis or to start and stop the automatic processing of the IPP.

2.3.1 ippMonitor

The `ippMonitor` is a web-based tool that allows the end-user to explore at a high level the results of the IPP analysis. There is an initial login page, after which the user is asked to choose a project. Once a project is selected, the user may view pages which show summary information of the images that have been processed at each of the different stages. There are top-level sections for the detrend analysis, the single science image analysis, and the different image combinations. Further clicking takes the user to more detailed information about the individual images: jpegs of the reduced image, plots showing the psf modeling and the astrometric analysis, etc.

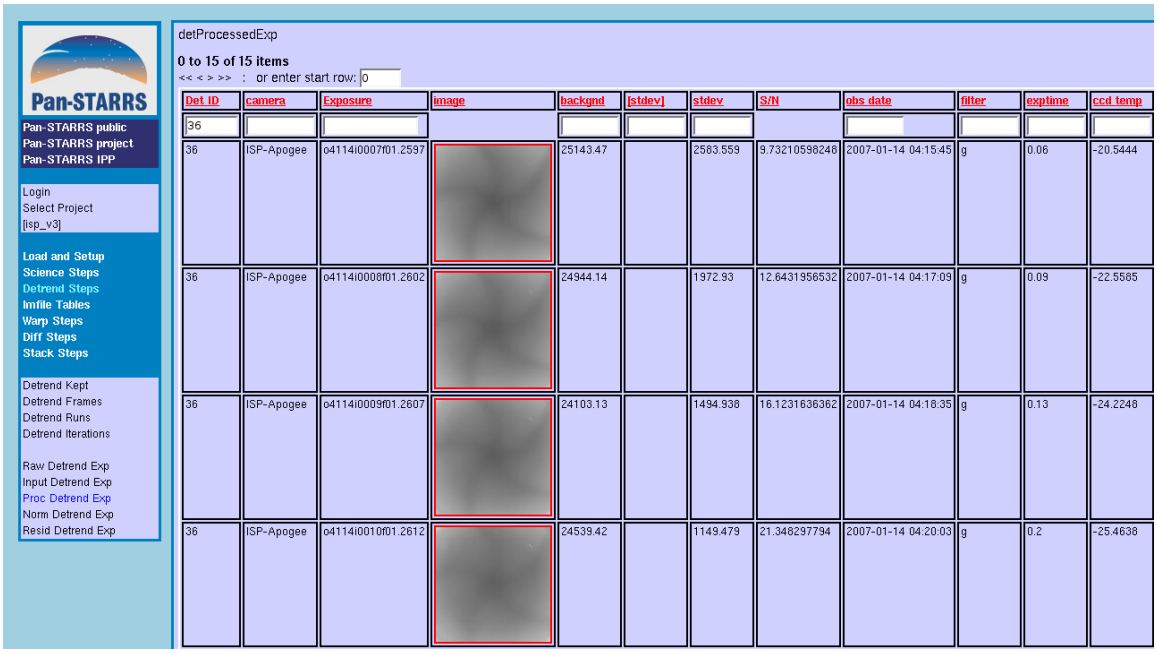

Figure 2: Example Screen Shot from the ippMonitor

to be added to ippMonitor: pages showing summary plots from DVO

2.3.2 DVO Shell

The DVO database is the primary output destination for results from the IPP. The DVO shell is the basic user interface to the DVO database. Within the DVO shell, the user may extract measurments from the database, create plots, or view the locations of objects and images on projections of the sky.

A complete user's guide is available at DVO REF. Here are a few quick-start details:

Start the DVO shell by typing: `dvo`. You will be given a prompt, at which you may type commands. This is an interactive shell, with standard readline comand-line editing features. The `help` command gives further information about the commands. Commands will give additional information if they are given a '-help' option, or in many cases if the command is typed without arguments.

```
catdir skycells
region -n view1 0 0 80 ait
images

catdir taurus-project
region -n view2 64.8 26.1 4.0
images
avextract ra,dec where (g - i > 2.0)
cplot ra dec -c red
avextract ra,dec where (g - i < 0.5)
cplot ra dec -c blue
```

Choose the DVO database with the command `catdir`. **you must specify the path to the top of the database directories**. Define a view port for plotting objects and images on a sky projection with: `region RA DEC range [projection]`. The projection may be `ait` (aitoff projection), `sin` (sin projection), ...

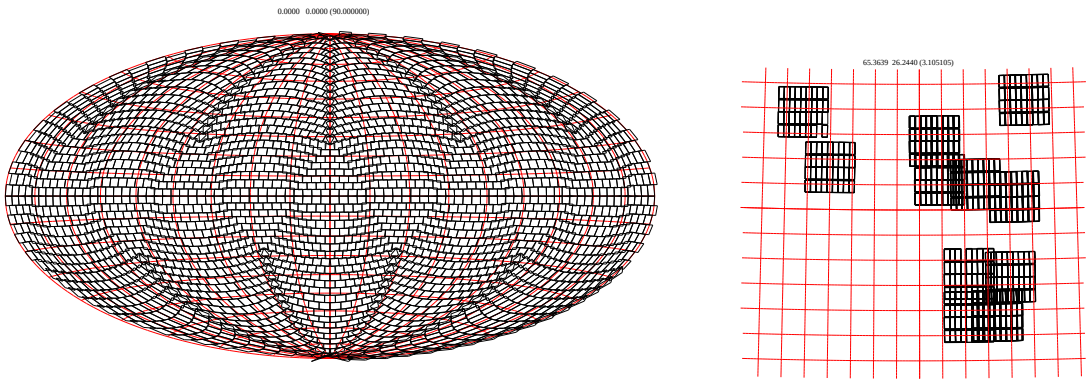

Figure 3: Example Plots from DVO

2.3.3 Pantasks

Pantasks is the program which schedules the analysis to be performed and distributes the analysis jobs across the parallel processing environemnt. A more complete user's guide is available at REF. Here we give a basic overview of starting up the IPP. In this example, we are using the stand-alone pantasks program. It is also possible to run pantasks in a client / server mode. In this mode, the server runs continuously in the background, and multiple users may interact with the same pantasks server via the pantasks client tool. Most of the commands are the same in both contexts. However, for security reasons in the client / server mode, the clients are not able to issue certain commands to the server. Thus, there are some differences in the details.

Start `pantasks`. Load the collection of IPP tasks with the command: `module pantasks.pro`. Add remote analysis hosts to the parallel processing controller (pcontrol): `controller host add NAME`, where NAME is the name of a machine set up to run the IPP. Add the IPP project(s) of interest: `add.database (project)`. Start the analysis: `run`. Check the current status with `status`. If something goes wrong, stop the processing with `stop` or `halt`. The former will continue to accept the results from the jobs already launched, while the latter halts all processing. **halt does not current stop pcontrol**.

2.3.4 ippTools & ippScripts

The IPP uses a `mysql` database to keep track of the data to be analysed and the results of analysis steps. The IPP user interacts with this database via a set of programs called `ippTools` and `ippScripts`. The difference between these is that the `ippTools` are somewhat lower-level programs to interact directly with the database, while the ippScripts perform a more complex operation, sometimes wrapping multiple calls to commands within `ippTools`.

- `ipp_serial_inject.pl` : inject data into the IPP database.
- `dettool -definebyquery` : define a new detrend run. With this command, the user defines a set of criteria which define a collection of input images to be used to generate a detrend image. The parameters which define the output detrend image also specified.
- `dettool -updatedetrun` : manually set the state of a detrend run.
- **extend this list...**

The ippTools accept a common set of command-line arguments which modify their behavior:

- `-pretend` : show the expected result, but do not apply the results to the database
- `-simple` return results with one line for each output component. Without this flag, the results are returned with the versbose, but self-documenting, psMetadataConfig format.
- `-dbname` specify the database to use
- **extend this list...**

The collection of programs which are used to initiate different stages of the IPP analysis is called `ippTools`. This suite of programs

2.4 Operational Scenarios

2.4.1 Reducing an Observing Run

For a single observing run, the user will typically have data from several nights, both science and raw detrend images. The following steps illustrate the analysis with data from a sample observing run. We assume the user has already installed the IPP software and the external software (mysql, apache, php, etc), set up the configuration for the camera, defined a project for this analysis, started pantasks, and loaded the analysis tasks. This example shows the steps to build the detrend data and process the science data.

The first step is to inject all of the data into the IPP database with the command `ipp_serial_inject.pl`. It is possible to define abstract data storage paths in the user's `.ipprc` file in the `DATAPATH` section. Each line gives the relationship between an abstract data location and the real-world UNIX path to that data. The script `ipp_serial_inject.pl` will interpret the location of the data in terms of the DATAPATH hierarchy when identifying the data for the database. When injecting the data, it is convenient to define a working directory with the `--workdir` option to `ipp_serial_inject.pl`. The science processing steps will use this location to store the output data products.

As the images are injected, they will be placed in the table of `new` exposures. If `pantasks` is running with the modules loaded, the images will be processed for registration. In this step, a minor analysis of the image statistics and an examination of the header information is performed, and the results used to provide further information about the images in the database. As images are registered, they migrate from the `new` to the `raw` tables. These tables can be view in the `ippMonitor` under the `Load & Setup` pages.

Once the images have been injected, the user will want to create, in order, bias, dark, shutter, flat, and fringe frames. As these are created, they may be used to generate the next level of image. It is important not to define, eg, a flat run before the lower-precedence elements are defined. At the moment, it is not possible to define a detrend analysis which blocks until a previous detrend analysis is complete. To start, define a bias based on a query to the database:

```
dettool -dbname name -definebyquery -det_type bias etc
```

For a small observing run, it is probably possible to include all bias images in the initial pass. IPP will iteratively reject complete input images which are outliers from the ensemble, as well as individual pixels. In the `ippMonitor`, rejected images will be greyed out in the residual pages. Examine the resulting residual images. If there are patterns to the residuals, it may be necessary to define subgroups, perhaps by time or ccd temperature. For example:

After a bias or set of biases are defined, follow the same process for a dark frame. It may be necessary to supply a non-linear dark current correction (perhaps one for each amplifier or chip). The script `ipp_darkstats.pl` can be used to examine this trend.

If a shutter correction is necessary, make sure to obtain data which can be used to generate the correction: a series of flat-field images with a range of exposure times, particularly near the short end of the range. If the shutter correction will be skipped, set the SHUTTER entries in the camera's ppImage.config file to FALSE.

A flat-field image is next. Initially, we must define a raw flat-field image, from dome or twilight flat-field images. Below, we will discuss the generation of a flat-field correction image from dithered science images. Do not forget to specify the filter for each flat-field image

Up to now, we have been ignoring the bad pixel mask. At this point, the user may generate a bad pixel mask from the collection of processed flat-field images. Use the script `UNKNOWN` to build a mask image. Turn on the use of the mask by setting `MASK` to true in the `ppImage.config` file. It is probably a good idea to now re-run the previous stages with the mask applied: this will result in better statistics for the residual images.

If certain filters require a fringe image, the user may inform the IPP of this fact by setting the FRINGE.FILTERS entry in the `ppImage.config` file. Use a set of night-time sky images to generate a fringe frame. **How to specify more than one fringe mode? (TBD)**

Activate the chip-level science analysis tasks in pantasks: chip.on. With this off, no data will be processed before the detrend images are ready. Once this is turned on, the science images will start to be processed.

Discuss blocks, re-queuing science images, analysis versions (TBD) .

Illustrate how to specify the DVO output database (TBD)

2.4.2 Pipeline Reduction : PI-Oriented Observatory

An PI-oriented observatory has the opportunity to operate the analysis system on a somewhat larger scale than the individual user. At a PI-oriented observatory, there will normally be a rotating suite of instruments, each of which is mounted on the telescope for a substantial number of nights. Over the course of months and years, the observatory will collect enough data from the instruments to perform a much more detailed level of analysis of the instrument than is possible in a single observing run by a single PI. In such a situation, the observatory may use the IPP to generate and track the high-quality master detrend images, to apply the detrend images to the data as it is being collected, and to monitor the instrumental zero points and other calibration information. The basic concepts of running the IPP at a PI observatory are identical to that of a single user. There are, however, a few modifications which may be interesting.

First, it is natural to automatically inject and register the data as it is obtained at the telescope. The telescope readout software could be modified to call `ipp_serial_inject.pl` after each image is obtained. Alternatively, this step could take place downstream wherever the data are archived. In an observatory environment, more so than in a single-user situation, it may be perfereable to use the Nebulous infrastructure to manage the location and copies of the image data. **Define an inject mechanism that uses Nebulous (TBD)** .

Second, it is interesting to consider on what timescale to test and re-build the master detrend images. In the single user case, the possible input detrend images are fairly limited. As illustrated above, the single user may use the IPP to define master detrend images and test the consistency of the inputs against these masters. In an observatory environment, it may be more appropriate to build the masters only on regular intervals when there is a suspicion the instrument may have changed. The periods in which the instrument is mounted define an obvious possible starting point for this period. Alternatively, the detrend creation steps in the IPP can be defined to run in verify mode on a nightly basis to test the

current state of the instrument. For the PI-oriented observatory, the number of variables may make it difficult to rely on these results in an automatic fashion; user examination of the results with `ippMonitor` may be needed to decide if a new master should be built.

2.4.3 Pipeline Reduction : Survey Observatory

A telescope used as a survey instrument has a somewhat different, more limited range of circumstances, than a PI-oriented observatory. There is generally a pre-defined schedule, perhaps with only a single instrument. The instrument configuration is more controlled, and the types of observations are more consistently defined. In this case, an automatic validation process can be more reliably used. The IPP can be set up to run the detrend validation stage at the beginning of every night, to automatically generate a new master if any of the input detrend images are out of spec, and to start automatic processing of the night's data with the new master images when they are ready.

3 IPP Components

Need to document command-line arguments, perhaps even algorithms. (TBD)

3.1 Analysis Programs

3.1.1 psphot

The detection and characterization of astronomical objects within an image is performed by the IPP component called `psphot`. This software may be used as a stand-alone program, or it may be called by other IPP programs to operate on the images they generate. A more detailed guide is available at REF. Here we summarize the basic operation and command-line options.

The basic call to the stand-alone version of `psphot` is: `psphot -file input.fits output [options]` where `[options]` represents options to the IPP configuration system as well as some specific to `psphot`. In the case of a "split" camera format, where separate chips or amps are stored in separate files, the "input.fits" may be a UNIX file glob which will expand to the set of file; note that the glob must be protected with double quotes for psphot. Alternatively, the input file list may be written to a text file and the `-file input.fits` entry replace with `-list file.txt`. The output entry is used as the root of a number of possible output files selected by the user.

Standard IPP-wide command-line options (document elsewhere):

- `-site site.mdc`
- `-camera camera-name`
- `-recipe NAME VALUE`
- `-D KEY VALUE`
- `-Df KEY VALUE`
- `-Db KEY VALUE`
- `-Di KEY VALUE`

Here are other possible command-line options specific to psphot:

- -version : report version info and exit
- -mask mask.fits
- -weight weight.fits
- -modeltest X Y : run the object model on the object at the given coordinates
  - -model
  - -fitmode
  - -fitset
- -photcode : manually specify the photcode for this image
- -region : limit the analysis to a specific image area
- -chip : limit the analysis to the given list of chips
- -psf : use the specified psf model (do not build a psf model)
- -src : perform photometry on the positions given in the list

There are many configuration options in the `psphot.config` recipe file. These are defined in the `psphot` manual.

3.1.2 psastro

3.1.3 ppStats

3.1.4 ppImage

3.1.5 ppMerge

3.1.6 ppNorm

3.2 Pipeline Infrastructure

3.2.1 ippdb / dbconfig

The IPP uses a mysql database to track the data to be processed, the current state of the processing steps, and summary information from each of the analysis steps. The database schema is defined by the set of MDC files in `dbconfig`. These files are used to generate a set of C-level APIs to access the database tables.

4 Configuration

Correct use of the IPP depends on a collection of configuration files. These files define information about the processing environment, such as the location of reference data or the name of computers available for analysis or storage. They also

define the operating parameters for the different programs, and choices about which analysis steps to perform.

Configuration information for the image processing is provided on four levels: the site-specific information, camera-specific details, information related to different formats of data from the same camera, and recipe configurations for the different analysis programs or steps. The configuration information is stored in files using the "psMetadataConfig" (MDC) file format, which is briefly described below (§4.0.5); for a more detailed description, see the psLib SDRS (PSDC-430-007).

The configuration levels for the image processing components of the IPP are:

- Options for the particular installation of the IPP: the *site*;
- Options specifying the details about the instrument: the *camera*;
- Options specifying the format of the FITS file: the *format*; and
- Options specifying the particular parameter choices that affect the details of an analysis: the *recipe*.

Note that these are arranged in an hierarchical order, with the site configuration being the most general, and the recipe configurations the most specific. For example, not all sites will have to deal with all cameras, and different cameras may require different recipes at different times according to their particular quirks, analysis experimentations, or their evolution.

We have provided examples of each of these configurations in the `config` component of the IPP, which should be a useful guide for setting up your own. The Pan-STARRS IPP Configuration Guide (PSDC-430-???) has all the detailed information.

4.0.4 Setting up configuration files

When the IPP software is installed, it generates a copy of the top-level *site* configuration file: `PREFIX/share/ippconfig/ippr`
Programs look for this information in the user's home directory at `~/.ipprc`. A new user may want to link `~/.ipprc` to the installed copy `PREFIX/share/ippconfig/ipprc.config`, in which case new installations will be immediately available. Alternatively, the user may want to copy the file in order to make their own modifications, if for example the user needs to modify the recipes for a specific camera.

After the *site* configuration file is read by IPP programs, they then attempt to read the *camera* configuration files defined in the *site* file by the list of cameras. These entries refer to files for each camera. The IPP configuration system searches for these files (and others referred there in) in the PATH defined at the top of the *site* file. This value mimics the UNIX PATH variable, and consists of a colon-separated list of locations. The first entry found from this list is used by the configuration system, allowing a user to override, for example, the globally installed camera configuration for some camera.

4.0.5 Overview of MDC format

psLib defines a `psMetadata` structure which can carry labeled data of arbitrary types. Originally designed for carrying the data in FITS headers, the `psMetadata` have proved so generally useful that we use them for our configurations (and a multitude of other uses!). We have designed a human-readable text-based format — the "MetaData Configuration" (MDC) format — which we use for this end.

Each simple entry in an MDC file must contain the name, type and value. Each of these is on a single line, separated by whitespace, and in that order. Comments may be placed at the end of the line (or on a blank line), after a hash mark (#). Whitespace at the beginning and end of strings (either the name, value or comment) are stripped.

The simple types follow the psLib types. Integers are specified by a letter indicating if the integer is signed (`S`) or unsigned (`U`) and a number indicating the dynamic range in bits (8, 16, 32 or 64); e.g., `U8` is commonly used for bit mask values, `S32` is commonly used for ordinary integer values. Floating point values are specified by the letter `F` and a number indicating the precision in bits (32 or 64): `F32` (single precision) or `F64` (double precision). Strings are specified by `STR`. Times may be specified with the following types: `UTC,UT1,TAI,TT`; values for the time are expected to be in ISO8601 format (`YYYY-MM-DDTHH:MM:SS.sZ`).

Names are traditionally all-caps, though there is no reason why they must be; the names are case-sensitive. A name may not be repeated unless it has previously been declared to be of type `MULTI` (no value should be provided with this declaration):

```
COMMENT    MULTI
COMMENT    STR    Having more than one COMMENT like this
COMMENT    STR    is permitted because of the MULTI.
```

A hierarchy can be made using the `METADATA` type, which signals a new level:

```
JANITOR    METADATA
    NAME          STR    John Doe
    PAY           F32    1234.56
    ECCENTRICITY  STR    9.87
END
```

Note that a `METADATA` block is closed by an `END`. No identing need be done within a `METADATA` block, but it is useful to be able to see the levels at a glance (just like in a C program). `METADATA` blocks may be nested within `METADATA` blocks, probably down as far as you have the patience to try. Note that `MULTI` declarations only apply to the current level — there is no inheritance.

The above format can be long if there are many `METADATA`s with similar contents. For this reason, we provide the `TYPE` declaration, which generates a `METADATA` with the contents each of type `STR`:

```
TYPE          EMPLOYEE    NAME       PAY        ECCENTRICITY
```

Now, the type `EMPLOYEE` may be used, with string values (NB: no spaces allowed!) to specify multiple entries:

```
JANITOR       EMPLOYEE    JohnDoe    1234.56    9.87
PROGRAMMER    EMPLOYEE    FooBar     2345.67    1.00
```

This is the same as the much longer block:

```
JANITOR       METADATA
    NAME            STR     JohnDoe
    PAY             STR     1234.56
    ECCENTRICITY    STR     9.87
END
PROGRAMMER    METADATA
    NAME            STR     FooBar
    PAY             STR     2345.67
    ECCENTRICITY    STR     1.00
END
```

Like the `MULTI`, `TYPE` declarations only apply to the current level.

4.1 Filenames : UNIX Paths, Abstract Paths, Nebulous

The IPP programs recognize three types of file names: a standard UNIX path, an abstract path with a top-level location defined by the IPP configuration system, and a Nebulous path, in which the file location is completely abstracted by the Nebulous file management system:

- `file:///path/to/file.ext` : a UNIX path, always relative to the root of the file system. It is valid to drop the `file:/` component of the name. The IPP accepts an arbitrary number of slashes after `file:`.
- `path://PATH/file.ext` : An abstract path. The value of PATH is iterpolated from the corresponding entry in the `DATAPATH` table given in the site configuration file.
- `neb://full/nebulous/name` : A nebulous filename. The IPP programs query Nebulous for the true path of (one copy of) the file.

If you wish to expand a file name in any of these forms to a standard UNIX path, use the program `ipp_datapath.pl`.

5 Installation

5.1 psconfig

The psconfig system allows the user to build and install the IPP software suite into a location which is flexibly defined by the user. The tools here also set up the user's environment variables (`PATH`, `PERL5LIB`, `LIBRARY_PATH`, etc) to make use of the installed software. With the psconfig tools, it is easy to switch between different installed versions or to recompile subsets of the IPP tree. It is also possible for a user to install the complete IPP system without access to the root password.

5.1.1 Preparation

5.1.1.1 Unpack the IPP tarball.

The IPP source code is distributed as a single tarball for all of the IPP software (ipp-M.NN.tgz, where M.NN is the version number, currently 2.1). Use the command `tar xvzf ipp-M.NN.tgz` or `gzcat ipp-M.NN.tgz | tar xvz` on older platforms. The tarball will unpack into a directory named ipp-M.NN.

5.1.1.2 Unpack the external Perl modules.

If needed, the complete collection of Perl modules used by the IPP is also distributed from the IPP web site as a tarball, extperl.tgz. Use the command `tar xvzf extperl.tgz`, which will unpack the tarball into a directory called `extperl`. This must be at the same location as the top level directory. The program `pscheckperl` below will identify any external Perl modules which are missing from your system.

5.1.1.3 Download the external C libraries.

If needed, the external libraries needed by the IPP are available from the IPP web pages. Download these as needed to a directory called `extlibs`, again parallel to the `ipp-M.NN` directory. The program `pschecklibs` below will identify any external C libraries which are missing from your system.

5.1.1.4 Set up your account to use the psconfig system.

The psconfig system provides a script which sets up the necessary UNIX environment variables and aliases, enabling the compiler and the programs to find your installed libraries and Perl modules. The details depend on your UNIX shell:

5.1.1.4.1 csh users

To use the psconfig system, place the following line in your /.cshrc file:

```
alias psconfig "source ../ipp-M.NN/psconfig/psconfig.csh"
```

where ../ipp-M.NN is the location of the extracted ipp tarball. If you prefer, you may copy the file psconfig.csh to another location. If, for example, you would like to remove the build directories after building the IPP, you will need a persistent copy of psconfig.csh.

By default, the psconfig system places the installed IPP programs and configuration files in a directory in your home directory: `~/psconfig`. To use a different location, place the following line in `~/.psconfigrc` (otherwise not needed):

```
set PSCONFDIR = INSTALL_PATH
```

where `INSTALL_PATH` is the top-level directory for all installed files.

5.1.1.4.2 bash users

Add the following line to your `~/.bashrc` file:

```
alias psconfig="source PATH/psconfig.bash"
```

It is also necessary to edit the file psconfig.bash to set the `PSCONFIG_DIR` variable to this directory as well.

set this up for better discovery (TBD)

5.1.1.5 Set the installation version

The IPP is a large and complex software system. A major goal of the IPP build system is to be user-friendly for those end users which do not have root access on their machines. Using the IPP build tools, it is possible to install the complete system as a non-priviledged user. The build system also makes it possible to maintain multiple simultaneous installations with different versions of the software. This latter feature is particularly important for developers who need to be able to make tests and comparisons of different versions.

With psconfig, you may have multiple, parallel installations of the IPP. These may be different versions of the software, or installations for different computer architectures, or installations with different compilation options. For example, you may want an installation with tracing and debug information turned on and a second with all optimizations turned on. With the psconfig build system, each installation of the IPP software is placed in a directory identified by an installation name and the computer architecture. The installation name is up to the person who compiles the IPP software, and is an arbtrary word or string. For example, you may choose the install the optimized version under ipp-2.1-opt and the debug version under ipp-2.1-debug. The psconfig system will create a directory in ~/psconfig (or where ever PSCONDIR is set) called ipp-2.1-opt.linux (if building on a linux 32bit system), or something equivalent.

Before running or compiling the IPP, it is necessary to use psconfig to set the installation name: psconfig (name) This command sets aliases and environment variables for the current shell to point at the named IPP installation, for the current hardware. For example: psconfig default will set the PATH to include ~/psconfig/default.linux/bin on a 32-bit linux system, and the other paths to point at the corresponding installation directories.

Users who wish to automatically have access to an IPP installation should add the psconfig line to their ~/.cshrc or ~/.bashrc files.

5.1.2 Check External Dependencies

The IPP build system is run from the directory ../ipp-M.NN/psconfig. Within this directory are the build scripts and scripts to prove the build environment. Before building the IPP suite, you should first check for the needed C libraries and Perl scripts. Start by cd-ing into ipp-M.NN/psconfig.

5.1.2.1 External C libraries

The program pschecklibs in the ipp-M.NN/psconfig directory will check for required system libraries and headers: pschecklibs It examines the system libraries, libraries defined by LIBRARY_PATH, and the installation library defined by psconfig. Any missing dependencies will be listed. Tarballs for these libraries may be found on the Pan-STARRS web site at: http://pan-starrs.ifa.hawaii.edu/project/IPP/software/ext These should be installed so they will be available in the user's path. This can be done with the psconfig tools by using psconfigure to replace the standard configure command. The psconfigure command applies the needed --prefix options to place the libraries within the IPP installation tree. There is also an equivalent for autogen.sh, psautogen, if needed.

5.1.2.2 External Perl Modules

The program pscheckperl in the ipp-M.NN/psconfig directory will check for required Perl modules, and can be used to install them in the appropriate user location in the psconfig system. The command defaults to the latest perl installation table in the tagsets directory. If you have CVS access and choose to check out an older IPP distribution, it is necessary to supply the distribution name to pscheckperl. Otherwise, run it without arguments: pscheckperl This will test for the perl modules specified for the latest ipp release. If any modules are missing, you should install the perl module tarball (see above), or they can be download from the Pan-STARRS web site: http://pan-starrs.ifa.hawaii.edu/proj

The tarballs should be placed in a directory extperl parallel to the ipp directory (two levels up from this directory). If the tarballs are in the correct location, they can be built by supplying the -build flag to pscheckperl:

```
pscheckperl -build
```

5.1.3 Building IPP

To build the full IPP tree using the psconfig system, run `psbuild` in the `ipp-M.NN/psconfig` directory. By default, `psbuild` will use the latest IPP distribution table, in the directory `tagsets`, and build the IPP components according to the rules in that distribution file. The distribution files specify which versions (which CVS tags) of the different programs are to be built, and in which order, for a given distribution of the IPP. If you have access to the IPP CVS tree and check out an old IPP distribution, it is possible to specify the file for that distribution, and build the software as it appeared for that older distribution. It is also possible to use the tools in this directory to check out the older code and to build tarballs for the older distributions. However, for the typical end user building the IPP from a distributed tarball, it is only necessary to run `psbuild` without additional arguments: `psbuild ipp-1.2`

There are a number of command-line options to `psbuild` which control how the software is built or which components of the IPP to build:

```
-version (version) : specify alternate psconfig installation version
-clean             : clean the source directories before building
-rebuild           : run 'autogen' (C code)
-optimize          : set flags for optimized code
-only (module)     : only build the specified module
-start (module)    : begin build at specified module
-stop (module)     : stop build after specified module
```

Summary of psconfig operations:

```
psdist -tag        : tag CVS tree
psdist -dist       : build tarball from tagged tree
psdist -dist -head : build tarball from head
psbuild            : build and install software in tree
pschecklibs        : check for needed external software
pscheckperl        : check for needed perl modules
pscheckperl -build : build and install external modules
```

5.2 jhbuild

JH uses `jhbuild` even though the 'jh' in `jhbuild` doesn't really refer to him.

5.2.0.1 What is it?

According to the introduction on the `jhbuild` website:

> `jhbuild` is a program that can be used to pull a number of modules from CVS and build them in the correct order. Unlike some build scripts, `jhbuild` lets you specify what modules you want built and it will then go and build those modules plus dependencies.
>
> Although `jhbuild` was originally developed to build `[WWW]Gnome`, it is now able to build a number of the modules in `freedesktop.org` CVS. Extending it to handle new modules is usually trivial (assuming the build infrastructure matches the other modules it handles).

In additional to retrieving source code from various SCM's (CVS, SVN, arch, etc.), jhbuild has the ability to download tarballs via HTTP or FTP.

jhbuild has been adopted as an official freedesktop.org project. You can find more information on the project's homepage (http://www.freedesktop.org/Software/jhbuild). Bugs can be filed in the Gnome Bugzilla (http://bugzilla.gnome.org).

5.2.0.2 Where to get it

It was necessary to slightly modify jhbuild for use with IPP software. Therefore, you must checkout the jhbuild module from the Pan-STARRS CVS tree. Please see the Pan-STARRS CVS Guide for help on setting up and using CVS. jhbuild will need to be able to find it's own source tree even after installation so you should choose a checkout path that can be permanent. Something along the lines of $HOME/src is recommended.

```
cd
mkdir -p src
cd src
cvs co jhbuild
```

After running CVS you should see something like this:

```
$ cvs co jhbuild
cvs checkout: Updating jhbuild
U jhbuild/.cvsignore
U jhbuild/COPYING
U jhbuild/ChangeLog
U jhbuild/HACKING
U jhbuild/Makefile
U jhbuild/README
U jhbuild/install-check.c
.
.
```

5.2.0.3 Installing jhbuild into your home directory

jhbuild should be installed locally under your home directory. This will require that you modify the PATH environment variable so that you can run jhbuild after it has been installed.

```
cd jhbuild
make
make install
```

Which should look something like this:

```
$ make
gcc -Wall -O2 -o install-check install-check.c
Run "make install" to install.
$ make install
Creating /home/moanui/jhoblitt/bin/jhbuild
Creating /home/moanui/jhoblitt/.gnome2/vfolders/applications/jhbuild.desktop
install -m755 install-check /home/moanui/jhoblitt/bin/install-check
install -m755 config.guess /home/moanui/jhoblitt/bin/config.guess
```

That will install the jhbuild executable under $HOME/bin. You are responsible for including this path in your PATH environment variable. It is highly recommended that you add this to your .bashrc or equivalent shell login script.

For the bash shell, place this line in your .bashrc:

```
export PATH=${HOME}/bin:${PATH}
```

For the `tcsh` shell, place this line in your `.tschrc`:

```
setenv PATH ${HOME}/bin:${PATH}
```

5.2.0.4 Configuring jhbuild

`jhbuild` is configured via an rc file that lives at `$HOME/.jhbuildrc`. Please note that this rc file is executed as Python code; be careful!

Example `.jhbuildrc`, suitable for cut and paste:

```
# what profile to build?
moduleset = 'http://pan-starrs.ifa.hawaii.edu/project/IPP/software/modulesets/ipp12.modules'

# modules to build by default
modules = [ 'pslib', 'psmodules' ]

# where should working copies go?
jhroot = os.environ['HOME'] + '/jhroot'

# where should tarballs be kept?
tarballdir = jhroot + '/src'

# in what prefix should things be installed? (must be writable)
target = os.popen('config.guess').read().rstrip()
prefix = jhroot + '/' + target
checkoutroot = prefix + '/build'

# extra arguments to pass to the autogen.sh script?
autogenargs = '--enable-maintainer-mode --disable-static'

# use an alternative install program that preserves the
# mtime on header files if they haven't changed.  Speeds
# up rebuilds.
os.environ['INSTALL'] = os.environ['HOME'] + '/bin/install-check'

# don't try to use /usr/ucb/cc on Solaris
import sys
if sys.platform == 'sunos5':
    os.environ['CC'] = 'gcc'
```

5.2.0.5 Running jhbuild

`Jhbuild` can be executed as `jhbuild build [modulename]`. Just `jhbuild` will build the packages specified in the `modules` variable from your rc file.

```
jhbuild
```

or

```
jhbuild build pslib
```

Run `jhbuild` list to get a list of the packages `jhbuild` knows how to build.

```
$ jhbuild list
cfitsio
gsl
fftw
libxml2
mysql
pslib
psmodules
```

`jhbuild` supports many other commands. Please see `jhbuild --help` for a complete list of options.

```
$ jhbuild --help
usage: jhbuild [ -f config ] command [ options ... ]
Build a set of CVS modules (such as GNOME).

Global options:
  -f, --file=CONFIG            use a non default configuration file
  -m, --moduleset=URI          use a non default module set
      --no-interact            do not prompt for input

Commands:
  gui                          build targets from a gui app
  update                       update from cvs
  updateone modules            update a fixed set of modules
  build [ opts... ] [modules]  update and compile (the default)
  buildone [ opts... ] modules build a single module
  tinderbox [ opts... ]        build non-interactively with logging
  run program [ args... ]      run a command in the build environment
  shell                        start a shell in the build environment
  sanitycheck                  check that required support tools exists
  bootstrap                    build required support tools
  list [ opts ... ] [modules]  list what modules would be built
  dot [ modules ]              output a dot file of dependencies suitable
                               for processing with graphviz
  info modules...              prints information about modules

Options valid for the build, buildone, tinderbox and update commands:
  -s, --skip=MODULES           treat the given modules as up to date
  -t, --start-at=MODULE        start building at the given module
  -D date_spec                 set a sticky date when checking out modules

Options valid for the build, buildone and tinderbox commands:
  -a, --autogen                always run autogen.sh
  -c, --clean                  run make clean before make
  -n, --no-network             skip cvs update

Options valid for the tinderbox command:
  -o, --output=DIR             directory to save build logs in

Options valid for the list command:
  -r, --show-revision          show which revision will be built
```

5.2.0.6 Dependancies

`jhbuild` has a fairly minimal set of dependencies — far less than what may be required to actually compile and install any packages. However, if your system can meet the base requirements, `jhbuild` should be able to bootstrap your build environment.

- A working C compiler (eg. `gcc`)
- A working `libc` (eg. `glibc`)

- Perl 5 with the `XML::Parser` module (needed by `libtool`)
- Python 2.?
- Either `wget` or `curl`
- GNU `M4` 1.4
- `tar`
- `gzip`
- `bzip2`

5.2.0.7 Bootstrapping

`jhbuild` has a limited ability to install some of the necessary tools for maintaining software that configure its build environment with the GNU autotools.

This step is probably required on OSX and Solaris. Your mileage will vary per Linux distribution but you can probably skip this step if your distribution is around RedHat 9 vintage or newer.

```
jhbuild bootstrap
```

`jhbuild` will then will begin to build a series of packages.

5.2.0.8 Using the jhbuild enviroment

As you've already seen, `jhbuild` is capable of setting up an independent build environment under the (configurable) directory of your choice. In order to link non-`jhbuild` management software against this build environment a number of your shell's environment variable have to be modified. `jhbuild` is capable of doing this for you. The syntax for this is `jhbuild shell`, which as the syntax implies, spawns a new shell with the proper environment variables.

This example demonstrates `jhbuild` setting up the dynamic linkers default search path for you.

```
$ echo $LD_LIBRARY_PATH

$ jhbuild shell
$ echo $LD_LIBRARY_PATH
/home/moanui/jhoblitt/jhroot/i686-pc-linux-gnu/lib
```

A fair number of other variables are also adjusted for you. Enough so that most (all?) `autoconf` configured software will be able to find it's dependencies.

5.3 Aliases

PAP puts the following in his `~/.tcshrc`:

```
setenv SWDIR $HOME/local/‘$HOME/bin/config.guess‘/
if (! -d $SWDIR) mkdir --parents $SWDIR
alias ./autogen.sh ’./autogen.sh --prefix=$SWDIR CFLAGS="-I$SWDIR/include/ -g" LDFLAGS=-L$SWDIR/lib/’
alias   autogen.sh ’./autogen.sh --prefix=$SWDIR CFLAGS="-I$SWDIR/include/ -g" LDFLAGS=-L$SWDIR/lib/’
alias ./configure  ’./configure  --prefix=$SWDIR CFLAGS="-I$SWDIR/include/ -g" LDFLAGS=-L$SWDIR/lib/’
alias   configure  ’./configure  --prefix=$SWDIR CFLAGS="-I$SWDIR/include/ -g" LDFLAGS=-L$SWDIR/lib/’
setenv PATH ${PATH}:$SWDIR/bin/
setenv LD_LIBRARY_PATH $SWDIR/lib/:$SWDIR/lib/mysql:$LD_LIBRARY_PATH
setenv MANPATH $SWDIR/man:$MANPATH
setenv PKG_CONFIG_PATH $SWDIR/lib/pkgconfig/:$PKG_CONFIG_PATH
```

Here, `config.guess` is the common GNU script for guessing the build system triplet (e.g., `i686-pc-linux-gnu`).

There are a couple of notes:

- To compile a binary, simply do `./configure`, then `make && make install`.
- `Ohana` doesn’t like this setup, so you need to build it with: `\./configure --prefix=$SWDIR`
- Perl modules can be installed: `./Build install   --prefix=$SWDIR`.

5.4 Manual Perl Module Installation

Here we describe setting up the Perl dependencies followed by the IPP components.

5.4.1 Dependencies

If you have access to the `root` account, installation as `root` is much easier. If not, you will have to go through the more flaky installation as an unprivileged user.

5.4.1.1 Installation as root

Many of the Perl dependencies are available from the Comprehensive Perl Archive Network (CPAN) at www.cpan.org. If you have root access on your target machines, they can be very simply retrieved, built and installed (replacing `MODULE_NAME` for each module):

```
> su -
Password:
> cpan
[...]
cpan> install MODULE_NAME
[...]
cpan> quit
```

Follow the prompts. It’s usually safe to accept the default (simply hit enter) in response to most questions.

If you get into trouble, try: `force install MODULE_NAME`.

You can also try to use the `Bundle::PS` as described below if you’re feeling adventurous.

5.4.1.2 Installation as unprivileged user

To install modules from CPAN with the `CPAN.pm` interface, you need to setup a CPAN configuration file in your home directory. Then `CPAN.pm` can walk you through setting up the most important configuration values. Unfortunately, there is some variation in the behavior of the various versions of `CPAN.pm` that have shipped with Perl. Some (most) of these variants will not correctly create a configuration files that allows a non-`root` user to install modules outside of ”system” paths. In order to make sure that you get a ”correct” CPAN configuration file you need to “prime” it with a few values.

First you need to create the directory in which the CPAN configuration file will live.

```
> mkdir -p .cpan/CPAN/
```

Then we need to create a partial configuration file. Note that this example assumes that you want to install your perl modules under `$HOME/local/lib/perl5`.

```
> echo "\$CPAN::Config = {" >> .cpan/CPAN/MyConfig.pm
> echo "  makepl_arg => q[PREFIX=$HOME/local/]," >> .cpan/CPAN/MyConfig.pm
> echo "  mbuildpl_arg => q[--install_base $HOME/local/]," >> .cpan/CPAN/MyConfig.pm
> echo "};" >> .cpan/CPAN/MyConfig.pm
> echo "1;" >> .cpan/CPAN/MyConfig.pm
> echo "__END__" >> .cpan/CPAN/MyConfig.pm
```

Now you need to invoke `CPAN.pm` so it can walk you through configuring the rest of the required values. This is an example of one possible configuration with `CPAN.pm` version 1.8802. **Your version of CPAN.pm may present you with different prompts.** Use your common sense. If in doubt, it is generally safe to simply hit enter (and accept the default).

```
> perl -MCPAN -e shell
CPAN: File::HomeDir loaded ok
Sorry, we have to rerun the configuration dialog for CPAN.pm due to
the following indispensable but missing parameters:

build_cache, build_dir, cache_metadata, cpan_home, ftp_proxy, http_proxy,
index_expire, inhibit_startup_message, keep_source_where, make_arg,
make_install_arg, mbuild_arg, mbuild_install_arg, mbuild_install_build_command,
no_proxy, prerequisites_policy, scan_cache, urllist


The following questions are intended to help you with the
configuration. The CPAN module needs a directory of its own to cache
important index files and maybe keep a temporary mirror of CPAN files.
This may be a site-wide directory or a personal directory.



I see you already have a  directory
    /home/moanui/jhoblitt/.cpan
Shall we use it as the general CPAN build and cache directory?

CPAN build and cache directory? [/home/moanui/jhoblitt/.cpan]


Unless you are accessing the CPAN via the filesystem directly CPAN.pm
needs to keep the source files it downloads somewhere. Please supply a
directory where the downloaded files are to be kept. [/home/moanui/jhoblitt/.cpan/sources]
Directory where the build process takes place? [/home/moanui/jhoblitt/.cpan/build]
```

```
How big should the disk cache be for keeping the build directories
with all the intermediate files?

Cache size for build directory (in MB)? [100]


The CPAN indexes are usually rebuilt once or twice per hour, but the
typical CPAN mirror mirrors only once or twice per day. Depending on
the quality of your mirror and your desire to be on the bleeding edge,
you may want to set the following value to more or less than one day
(which is the default). It determines after how many days CPAN.pm
downloads new indexes.

Let the index expire after how many days? [1]


By default, each time the CPAN module is started, cache scanning is
performed to keep the cache size in sync. To prevent this, answer
'never'.

Perform cache scanning (atstart or never)? [atstart]


To considerably speed up the initial CPAN shell startup, it is
possible to use Storable to create a cache of metadata. If Storable
is not available, the normal index mechanism will be used.

Cache metadata (yes/no)? [yes]


The CPAN module can detect when a module which you are trying to build
depends on prerequisites. If this happens, it can build the
prerequisites for you automatically ('follow'), ask you for
confirmation ('ask'), or just ignore them ('ignore'). Please set your
policy to one of the three values.

Policy on building prerequisites (follow, ask or ignore)? [ask] follow


Every Makefile.PL is run by perl in a separate process. Likewise we
run 'make' and 'make install' in separate processes. If you have
any parameters (e.g. PREFIX, LIB, UNINST or the like) you want to
pass to the calls, please specify them here.

If you don't understand this question, just press ENTER.
Parameters for the 'make' command?
Typical frequently used setting:

    -j3             # dual processor system

Your choice:  []
Parameters for the 'make install' command?
Typical frequently used setting:

    UNINST=1        # to always uninstall potentially conflicting files

Your choice:  [] UNINST=1


The next questions deal with Module::Build support.

A Build.PL is run by perl in a separate process. Likewise we run
'./Build' and './Build install' in separate processes. If you have any
parameters you want to pass to the calls, please specify them here.

Parameters for the './Build' command?
Setting might be:
```

```
    --extra_linker_flags -L/usr/foo/lib  # non-standard library location

Your choice:  []
Do you want to use a different command for './Build install'?
Sudo users will probably prefer:

    su root -c ./Build
or
    sudo ./Build
or
    /path1/to/sudo -u admin_account ./Build

or some such. Your choice:  [./Build]
Parameters for the './Build install' command?
Typical frequently used setting:

    --uninst 1                          # uninstall conflicting files

Your choice:  [] --uninst 1


If you're accessing the net via proxies, you can specify them in the
CPAN configuration or via environment variables. The variable in
the $CPAN::Config takes precedence.

Your ftp_proxy? []
Your http_proxy? []
Your no_proxy? []
You have no /home/moanui/jhoblitt/.cpan/sources/MIRRORED.BY
  I'm trying to fetch one
CPAN: LWP::UserAgent loaded ok
Fetching with LWP:
  http://www.perl.org/CPAN/MIRRORED.BY


Now we need to know where your favorite CPAN sites are located. Push
a few sites onto the array (just in case the first on the array won't
work). If you are mirroring CPAN to your local workstation, specify a
file: URL.

First, pick a nearby continent and country by typing in the number(s)
in front of the item(s) you want to select. You can pick several of
each, separated by spaces. Then, you will be presented with a list of
URLs of CPAN mirrors in the countries you selected, along with
previously selected URLs. Select some of those URLs, or just keep the
old list. Finally, you will be prompted for any extra URLs -- file:,
ftp:, or http: -- that host a CPAN mirror.

(1) Africa
(2) Asia
(3) Central America
(4) Europe
(5) North America
(6) Oceania
(7) South America
Select your continent (or several nearby continents) [] 5

(1) Bahamas
(2) Canada
(3) Mexico
(4) United States
Select your country (or several nearby countries) [] 4

(1) ftp://carroll.cac.psu.edu/pub/CPAN/
(2) ftp://cpan-du.viaverio.com/pub/CPAN/
(3) ftp://cpan-sj.viaverio.com/pub/CPAN/
(4) ftp://cpan.calvin.edu/pub/CPAN
(5) ftp://cpan.cs.utah.edu/pub/CPAN/
```

```
(6) ftp://cpan.cse.msu.edu/
(7) ftp://cpan.erlbaum.net/CPAN/
(8) ftp://cpan.glines.org/pub/CPAN/
(9) ftp://cpan.hostrack.net/pub/CPAN
(10) ftp://cpan.llarian.net/pub/CPAN/
(11) ftp://cpan.mirrors.redwire.net/pub/CPAN/
(12) ftp://cpan.mirrors.tds.net/pub/CPAN
(13) ftp://cpan.netnitco.net/pub/mirrors/CPAN/
(14) ftp://cpan.pair.com/pub/CPAN/
(15) ftp://cpan.teleglobe.net/pub/CPAN
(16) ftp://cpan.uchicago.edu/pub/CPAN/
40 more items, hit RETURN to show them
Select as many URLs as you like (by number),
put them on one line, separated by blanks, hyphenated ranges allowed
 e.g. '1 4 5' or '7 1-4 8' [] 14 11 12

Enter another URL or RETURN to quit: []
New set of picks:
  ftp://cpan.pair.com/pub/CPAN/
  ftp://cpan.mirrors.redwire.net/pub/CPAN/
  ftp://cpan.mirrors.tds.net/pub/CPAN


Please remember to call 'o conf commit' to make the config permanent!


cpan shell -- CPAN exploration and modules installation (v1.8802)
ReadLine support enabled

 cpan[1]> o conf commit
commit: wrote '/home/moanui/jhoblitt/.cpan/CPAN/MyConfig.pm'
```

Now we need to install the module that installs the other modules.

```
cpan> install Module::Build
```

Exit out of cpan:

```
cpan> exit
```

In order to use of the installed modules, we need to setup an environment variable called `PERL5LIB` so that 'perl' can find them. To do this, we need to know where under 'perl5' our modules were actually installed. This will set variable with the version of Perl that you are using. The easiest way to do this is just just look in the root of the path where we did the install.

```
> ls local/lib/perl5/
5.8.8  site_perl
```

That means we're using perl 5.8.8 and `PERL5LIB` needs to be setup as following:

```
export PERL5LIB=$HOME/local/lib/perl5/5.8.8:$HOME/local/lib/perl5/site_perl/5.8.8
```

Now we should install the basic compliment of helper modules that `CPAN.pm` needs to function fully. Go back into CPAN (`perl -MCPAN -e shell`) and:

```
cpan> install Bundle::CPAN
```

You can quit out of the CPAN shell at this point with the ‘exit‘ command or do the following few steps in another shell We’re ready to install the full set Perl module dependencies for IPP software. In order to make this process a bit easier on the end user a ”Bundle” module has been created. In order to use it you need to create a directory (if it doesn’t already exist) called Bundle under your .cpan directory.

```
> mkdir -p .cpan/Bundle
```

The file `PS.pm` should copied into this directory:

```
cp /path/to/PS.pm .cpan/Bundle/
```

Back in the CPAN shell, ’force’ the install of the PS Bundle. The ’force’ keyword instructs the shell to ignore any tests failures. This is necessary as some of the modules ’DBD::mysql’/etc. require a properly working database setup in order for the tests to pass. You will most likely be prompted for input by several the modules. It is safe to answer with a carriage return to all questions. If it insists on a path to `httpd`, hit `CTRL-C` and it will go on to the next step.

```
cpan> force install Bundle:PS
```

For further instructions on installing Perl modules from CPAN ”by hand’, see:

```
http://www.cs.ucsc.edu/~you/notes/perl-module-install.html
```

After the dependencies (§6.2) have been satisfied, the IPP packages should be installed in the following order:

- `Ohana`
- `psLib`
- `psModules`
- `psphot`
- `psastro`
- `ppStats`
- `ppImage`
- `ppMerge`
- `ppNorm`
- `pois`
- `stac`
- `pswarp`
- `ppStac`
- `ippdb`

- ippTools
- PS-IPP-Metadata-Config
- PS-IPP-Metadata
- ippScripts
- ippTasks
- config

5.4.2 Modules and scripts

Pan-STARRS Perl modules and scripts are installed by executing the following in the directory containing the source:

```
> perl Build.PL
> ./Build
> ./Build install --prefix=/path/to/install/
```

Make sure you set your PERL5LIB environment variable to the installation path (here, /path/to/install/).

6 System Requirements

From the start, we have endeavoured to have the IPP to be flexible so that it can be deployed on a range of systems. With that in mind, however, we have specifically targeted Linux systems, stressing POSIX compliance.

6.1 Hardware

We have developed and tested on machines with different hardware architectures. Reports of successes and failures in deploying the IPP on an untested hardware architecture are welcome.

We have had thorough testing on x86 and amd64 architectures, with Gentoo and Red Hat Linux.

We expect that the binaries will compile under MacOS (if there are problems please let us know). We have had trouble compiling psLib under Solaris.

6.2 Dependencies

We have dependencies on two levels: library dependencies (for the binaries) and Perl module dependencies.

6.2.0.1 Libraries

The following external libraries are required to build psLib:

- gsl — http://www.gnu.org/software/gsl/

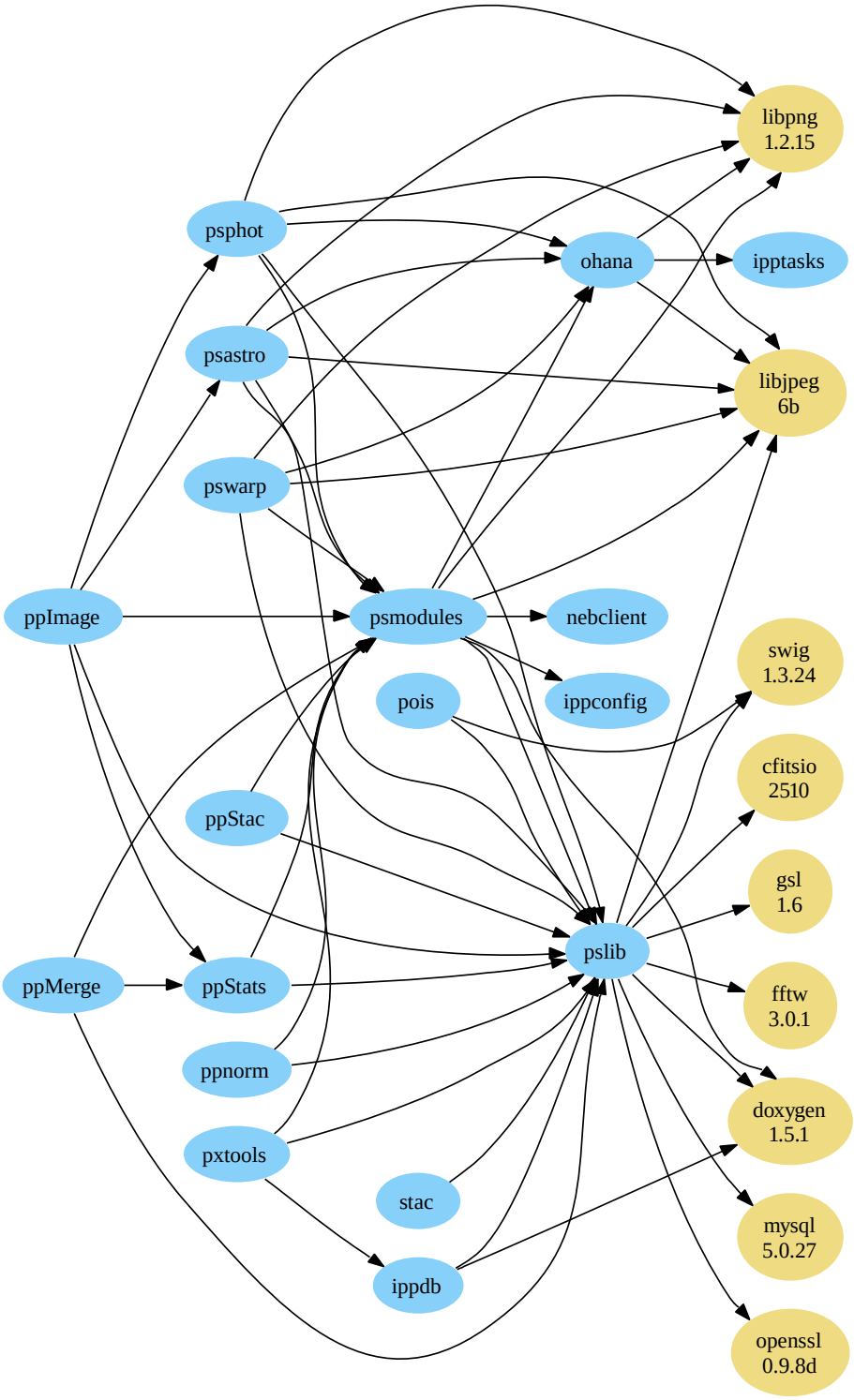

Figure 4: Dependency chart for the IPP binaries

- `fftw` — `http://www.fftw.org`
- `mysql` — `http://www.mysql.org`
- `cfitsio` — `http://heasarc.gsfc.nasa.gov/docs/software/fitsio/fitsio.html`
- `libjpeg` — `http://www.ijg.org`
- `openssl` — `http://www.openssl.org`
- `doxygen` (optional: for generating documentation) — `http://www.stack.nl/~dimitri/doxygen/`

In addition to the above, `libpng` (`http://www.libpng.org`) is required to build the `Ohana` package.

6.2.0.2 Perl modules

The following external modules, available from CPAN (see §5.4.1), are required to build the Perl scripts:

- `Module::Build`
- `ExtUtils::MakeMaker`
- `Params::Validate`
- `DateTime::TimeZone`
- `DateTime::Locale`
- `Time::Local`
- `DateTime`
- `MIME::Base64`
- `IO::Compress::Base`
- `Compress::Raw::Zlib`
- `Class::Factory::Util`
- `DateTime::Format::Strptime`
- `Net::Domain::TLD`
- `Sub::Uplevel`
- `HTML::Tagset`
- `Digest`
- `IO::Compress::Zlib`
- `version`
- `Text::Balanced`

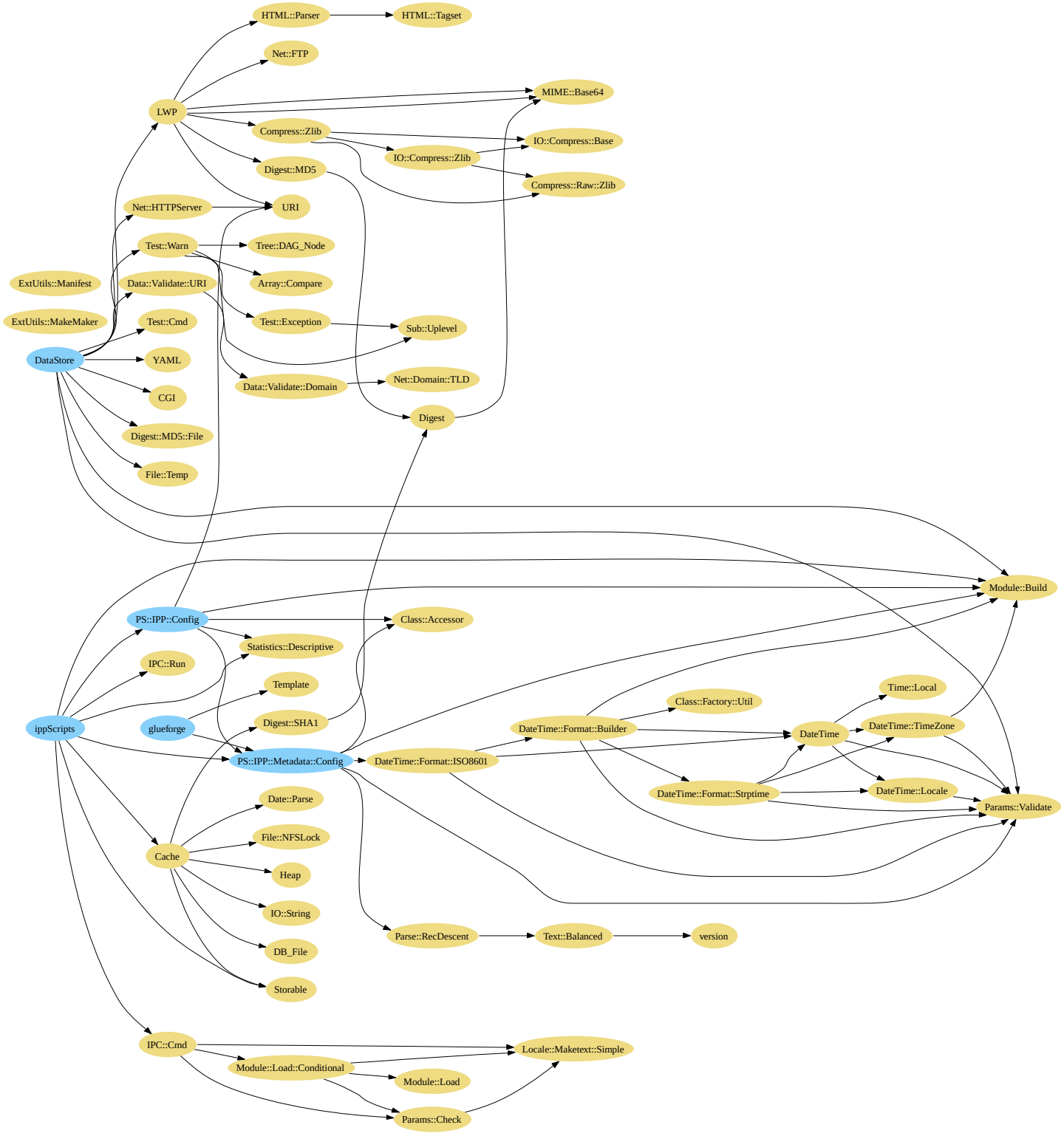

Figure 5: Dependency chart for the IPP Perl components

- DateTime::Format::Builder
- ExtUtils::Manifest
- URI
- Data::Validate::Domain
- Test::Exception
- Tree::DAG_Node
- Array::Compare
- HTML::Parser
- Digest::MD5
- Net::FTP
- Compress::Zlib
- Locale::Maketext::Simple
- Parse::RecDescent
- Class::Accessor
- DateTime::Format::ISO8601
- CGI
- Test::Cmd
- Net::HTTPServer
- Digest::MD5::File
- File::Temp
- Data::Validate::URI
- Test::Warn
- YAML
- LWP
- Module::Load
- Params::Check
- Template
- Statistics::Descriptive
- Storable

- `IO::String`
- `Date::Parse`
- `Digest::SHA1`
- `DB_File`
- `File::NFSLock`
- `Heap`
- `Module::Load::Conditional`
- `IPC::Run`
- `Cache`
- `IPC::Cmd`

6.3 Binaries

Installation of the binaries is complicated by the fact that they may be used on multiple architectures. The three developers based at Pan-STARRS HQ each use a different method for configuring the environment and installing the binaries to deal with this problem. We describe each of these below. Choose one that works for you!

7 Trouble Shooting Build Issues

7.1 missing libX11.so

RedHat RHEL 3 & 4 (likely other RedHat variants as well) for *amd64* seem to often be installed without a `libX11.so` under `/usr/X11R6/lib64/`. This will cause 64bit builds trying to link against libX11 (`-lX11`) to fail.

If you have root access you can fix this by adding the missing symlink yourself. Eg..

```
su -
cd /usr/X116/lib64
ln -s libX11.so.6.2 libX11.so
```

or with `sudo`:

```
cd /usr/X116/lib64
sudo ln -s libX11.so.6.2 libX11.so
```

For users without root access to thier system ,a symlink needs to be installed under `\$prefix/lib`, where `\$prefix` is the path underwhich your installing IPP software. Note that `\$prefix/lib` will also need to be manually added the enviroment variable `LD_LIBRARY_PATH` if your not using jhbuild. E.g for ksh/bash users:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/[foo]/[yourinstallprefix]/lib
```