

## **Where Will PyRAF Lead Us? The Future of Data Analysis Software at STScI**

P. Greenfield

R. L. White

*Space Telescope Science Institute, Baltimore, MD 21218*

**Abstract.** PyRAF, which allow the running and scripting of IRAF tasks from Python, was just the first step in STScI's migration to a more flexible and productive data analysis environment. The ultimate goal is to combine the best of the IRAF and IDL approaches to data analysis software into one environment, one where astronomers can find it easy to write software for, and software developers have sufficient tools and capabilities to handle writing general applications and pipelines. We are in the process of building the infrastructure to support these goals, and are well along in doing so. This paper will discuss the tools already developed, including PyRAF, numarray, PyFITS, numdisplay, and matplotlib, as well as the tools that are in development or are planned. Finally, it will review the applications that have been or are being developed and the implications for the user community of our move in this direction.

### **1. Introduction**

The PyRAF project was started to achieve both short-term and long-term goals. The short term goal was to provide a more robust and flexible scripting environment for IRAF tasks than the IRAF CL provided. At the time we were trying to maintain drizzle-related scripts that had gotten quite lengthy and were proving difficult to maintain. The absence of any error handling mechanism and easy identification of where errors occurred slowed progress considerably. An alternate CL looked like a promising way to overcome some of the existing limitations.

There were longer-range goals as well. These were not part of the initial development yet provided additional motivation for starting work on PyRAF. One was that it would enable easier integration of software not written under the IRAF environment with IRAF tasks. IRAF presents a comparatively closed software environment in that it is generally difficult to link other software tasks and libraries with IRAF tasks. While it is possible to run other software as "foreign" tasks from the IRAF CL, it still means invoking many separate processes and is a comparatively awkward way of integrating software. With a new CL, we could allow much easier integration of other libraries and be free of the restrictions that IRAF places on such software.

Finally, A new CL would enable the possibility of providing IDL-like data manipulation facilities, something that had long been desired by many users.

### **2. Why Python?**

We gave no thought to developing a new, custom scripting language. Developing and sustaining a good scripting language is a major effort. Past experience indicates that as-

tronomy is not a large enough field to warrant such an effort. Many scripting languages already existed and had sizable development communities.

In effect, one such 'scripting' language was already in wide use in astronomy, namely IDL. Granted, it isn't generally regarded as such, and although it isn't as general purpose as many of these, it shares at least a few characteristics with the better known ones to be viewed as one. Given its wide prevalence in astronomy, it was an obvious choice for trying as an alternate CL for IRAF. It's use would automatically satisfy the second of our long-term goals. Unfortunately, it lacked the necessary features to run IRAF tasks at the level of control we desired. This was the primary reason for rejecting it, though there were other important, but more secondary reasons. These included its cost, poor support for easily distributed C or Fortran extensions, and that it was not as general-purpose as other scripting languages, both in its language and library capabilities.

We deemed it important that any selected scripting language be popular enough, and have a bright enough future that it would be expected to be around many more years. At the time, there were three popular open source scripting languages: Perl, Tcl, and Python. The first, while very powerful and the most popular, was (and still is) considered difficult to learn, quirky, and led to code that was difficult to read, understand, and maintain. It generally is difficult to write large, maintainable programs in Perl. Tcl, on the other hand, has poor support for numerical types that rendered it unsuitable for astronomical contexts.

That left Python. It appeared to have all the necessary qualities including:

- Free, Open Source
- Very strong (deep and broad) user and development community
  - Still growing
  - Long lifetime expected
- General purpose
- Very portable
- Very extensible (in Python and with compiled code)
- Comparatively easy to learn
  - Yet very powerful
- Scales well

It was the clear choice at the time, and continues to be (<http://www.python.org>).

Initially, our goals were modest. We were not seeking to support IRAF graphics nor IRAF CL syntax. As the project progressed, we found that both these capabilities were within reach without enormous effort.

### 3. PyRAF Capabilities

Work on PyRAF began in 1998, and by mid-1999 it was being used internally at STScI. The first public beta release occurred in spring 2000. PyRAF provided a number of capabilities that were not present in IRAF at the time (a form of error handling and error traceback, as well as more conventional command line recall, have recently been added to IRAF, and are available in a beta release). The most notable aspect of PyRAF is that it doesn't require any changes to IRAF itself. It coexists with the existing IRAF CL; one can run one or the other once PyRAF is installed. PyRAF provides enhancements for interactive users and script writers.

For interactive users, the enhancements include:

- IRAF CL syntax accepted (as well as Python syntax)
- Ability to run most CL scripts
- Simple, up-arrow command line recall
- Tab completion of file names and commands
- Resizable GUI Graphics windows with plot recall
- GUI epar editor

For scripting, the following is possible:

- Full Python exception handling<sup>1</sup>
- Traceback information on error location
- Capture of task standard output to Python list structure

Not all aspects of the IRAF CL are currently supported. None of the current omissions are intrinsically not possible. Many are planned to be added in future enhancements (all those marked with \*). Some arise from the fact that the IRAF CL behavior is not well documented, and arise from user reports on discrepancies in behavior. These are noted and fixed as needed (more minor discrepancies remain and will be fixed in future releases). Some discrepancies will never be corrected as they highlight IRAF features that are best not emulated since they allow poor programming practices. Examples of these sorts of features include the ability to minimum match declared procedure variables (one may use `x` in place of `xcoord` if nothing else starts with `x`) or change variables outside of the scope of the procedure. In those cases we strongly recommend that IRAF CL scripts that use them be modified not to depend on such features (this is always possible and generally not difficult to do). The following list gives the major differences that currently remain.

- GOTOs not supported (next release will support forward GOTOs; already implemented).
- Background tasks\*
- Package unloading\*
- Text-based epar (for remote terminal use)\*
- Graphics redirection\*
- Emulation of new IRAF CL error handling\*

In addition to the currently marked differences between the IRAF CL and PyRAF, we plan many enhancements for PyRAF in the next year. Although we are no longer writing much software that requires PyRAF functionally (i.e., our new software no longer is written within the IRAF environment), it is still very important for our user community since it allows our new software to be run in an environment that is familiar to IRAF users and provides a nearly identical user interface. It also will play a key role in the Gemini calibration pipelines.

Further information may be found at:

[http://www.stsci.edu/resources/software\\_hardware/pyraf](http://www.stsci.edu/resources/software_hardware/pyraf)

---

<sup>1</sup>Note that the new error handling capability provided in IRAF ECL beta release does not provide exception handling in the usual sense. It executes the entire block of code and then jumps to the error handler at the end regardless of where the error occurs. Most exception handlers jump immediately on the error.

#### 4. Python's Larger Role

It is becoming well recognized by many software developers in astronomy (if not yet astronomers) that Python is a very good choice as a scripting language. Indeed, many of the existing or new astronomical software now sport, or will soon, some sort of Python interface. For example:

- Midas: PyMidas
- AIPS: Parseltongue
- ALMA
- CIAO/s-lang: PySL

The availability of Python interfaces to these systems will make it far easier to write scripts that combine capabilities from different systems.

Nevertheless, it is still a common viewpoint that Python is primarily only a scripting language and that it doesn't have a larger role to play in writing astronomical data analysis and reduction software. We have a more ambitious view. Our success in developing PyRAF led us to believe that it was a powerful language with the potential for being a primary, rather than secondary, language to develop applications in, provided the necessary tools were developed.

This was inspired, in part, by our previous experience with IDL that convinced us that it was possible to use an interpreted language to develop many useful applications. The presence of array manipulation facilities allows many computationally-intensive algorithms to be efficiently implemented in the interpreted language. While it is admittedly true that there are many scientific computation problems that are not well suited for array manipulations (e.g., fluid dynamics computations), our experience (and evidently the experience of many astronomers as well, given IDL's popularity in astronomy) was that such an approach was well suited to many data analysis and reduction problems.

No single language is suitable for all problems. Python is no exception. Yet, it is more sensible in our view to develop as much as possible in Python because of its highly productive nature than to take the more conventional approach of doing most development in a compiled language such as Fortran (any variant), C, C++, or Java, and then use a scripting language. The differences in productivity between the two approaches are startling, both in lines of code required and development time. Typically one sees increases in productivity ranging from factors 5 to 10 when switching from one of the traditional compiled languages to Python (much the same improvements have been seen by IDL users as well). Since Python provides very good interfaces to C (and thus to Fortran and C++), any part of the code that is too slow can instead call compiled code. Even with heavily computational code, generally speaking, most of the time spent in a program is usually confined to 10-20% of the code. Even if that code cannot be efficiently written in Python, it can be rewritten in C or Fortran. In other words: minimize the use of the less productive language.

An important side benefit of this approach is that it makes the code developed by professional software developers much more accessible to astronomers, and makes it easier for astronomers to contribute algorithms and utilities to a common software base. Currently, the two communities have little overlap in the software they develop; ending this divide will have many benefits.

#### 5. Enabling Python as a Data Reduction Language

Were the necessary facilities already available in Python? In part. There was an array package, Numeric (aka numpy) already available. However, it had a number of features

and characteristics that made it less desirable for dealing with the large data sets we see in astronomy. It did not have the facilities for making efficient use of memory, nor could it support an efficient use of table data structures. We also desired to use memory mapping to minimize memory usage. For these and other reasons we decided on developing a new array package, `numarray` ([http://www.stsci.edu/resources/software\\_hardware/numarray](http://www.stsci.edu/resources/software_hardware/numarray)).

Another essential library needed to support the use of Python for applications was a FITS module. This was initially developed by Paul Barrett and then adapted to use `numarray` and provide table access. Despite being written in pure Python, it is significantly faster for reading large files than the standard CFITSIO library, particularly for very large files. ([http://www.stsci.edu/resources/software\\_hardware/pyfits](http://www.stsci.edu/resources/software_hardware/pyfits))

Before Python could be considered a serious interactive environment for data analysis, it had to have a capable way of visualizing data. We developed `numdisplay` to display images to DS9 and `ximtool`. But for a long time, one of the biggest holes was the absence of a good plotting package that met all our needs (which included being free, open source, portable to all popular platforms, GUI-toolkit independent, image capable, and having high quality hardcopy). This was finally filled with the adoption of the plotting package `matplotlib` developed by John Hunter at the University of Chicago (<http://matplotlib.sourceforge.net/>).

Besides extending the capabilities of these existing tools, efforts are beginning on addressing more astronomically-specific libraries that contain utilities similar to those found in the IDL `astron` library to handle coordinate transformations, WCS coordinate systems, and photometry. Most of this work is taking place on a public repository to encourage contributions and enhancements from the general community.

([http://www.scipy.org/wikis/topical\\_software/AstroLib](http://www.scipy.org/wikis/topical_software/AstroLib))

Finally, there is currently an effort underway to unify the best aspects of the Numeric and `numarray` array packages into one called `scipy_core` (which will be as easy to install as as Numeric or `numarray`). We have done initial testing of `scipy_core` and it appears to satisfy all of the functional requirements we have. The effort to port our libraries and applications has begun. While `scipy_core` does not have an identical interface to `numarray`, it is much closer to `numarray` than Numeric in that regard. We will endeavor to make our software coexist with both during some reasonable transition period.

## 6. STScI Python Applications and Documentation

Most of STScI data analysis and reduction applications development is currently being done in Python. The following list summarizes those that have already been distributed or are completed and soon will be distributed. Most of these do not require IRAF at all (but many can be run as IRAF tasks in PyRAF using the standard IRAF CL syntax and `epar` interface). Most can be run on Microsoft Windows (though PyRAF cannot be because of its current dependence on IRAF).

- `PyDrizzle/Multidrizzle`: Drizzles a set of HST images onto a common image. `Multidrizzle` can remove cosmic rays even without any common pointings. (<http://stdas.stsci.edu/multidrizzle/>)
- `CALCOS`: The calibration pipeline for the Cosmic Origins Spectrograph instrument for HST.
- `FITSDIFF`: A utility for comparing FITS files for differences in the data or headers. (Distributed with `PyFITS`)
- Various STIS tools (Distributed with `stsci_python/STSDAS`):
  - `stisnoise`: removes fixed pattern noise

– `sshift`: aligns dithered spectral images

- `PySPV`: Python interface to Specview (Java GUI application)
- `NICMOS saaclean` tool: removes effects of SAA persistence (distributed with `stsci_python/STSDAS`)
- `TFIT` (from GOODS project): Does photometry on low resolution images using high resolution images for prior information.
- `WCS matching tool`: A GUI application for deriving WCS information for press release images from science images.

We have also developed a tutorial for astronomers showing how to use Python for interactive data analysis: [http://www.scipy.org/wikis/topical\\_software/Tutorial](http://www.scipy.org/wikis/topical_software/Tutorial)

## 7. The Future of Python at STScI

The more we use Python the more convinced we are that we made the right decision in 1998 to base our work on it. With the maturation of the tools for implementing data analysis algorithms directly in Python we anticipate a substantial increase in the number of astronomer-written scripts that are shared in the community. The STScI data analysis software group is committed to continuing support and development of our Python tools and applications, and we plan to help coordinate the distribution of user scripts and libraries as well.

**Acknowledgments.** The various projects described here have involved many contributions by several people:

- `PyRAF`: Richard L. White, Perry Greenfield, Michele De La Pena, Warren Hack, Philip Hodge
- `numarray`: Jay Todd Miller, Perry Greenfield, Jin Chung Hsu
- `PyFITS`: Paul Barrett, Jin Chung Hsu
- `numdisplay`: Warren Hack
- `matplotlib`: created and maintained by John Hunter at the University of Chicago with STScI contributions from Nadia Dencheva, Paul Barrett, Jay Todd Miller, Perry Greenfield
- `Regression Test Framework`: Howard Bushouse, Bernie Simon, Hemant Shukla
- `PyDrizzle/Multidrizzle`: Warren Hack, Christopher Hanley, David Grumm, Robert Jedrzejewski
- `CALCOS`: Philip Hodge
- `FITSDIFF`: Jin Chung Hsu
- `STIS tools`: Paul Barrett
- `NICMOS SAAClean`: Victoria Laidler
- `PySPV`: Ivo Busko
- `TFIT`: Victoria Laidler
- `Public Release WCS tool`: Philip Hodge