Development of the SOFIA Image Processing Tool

Alexander N. Adams
Dryden Flight Research Center
Major: Computer Engineering
USRP Summer 2011
Date: 19 AUG 11

Development of the SOFIA Image Processing Tool

Alexander N. Adams[*]
Richard Hang[†] (mentor)
Dryden Flight Research Center, Edwards, California, 93523

The Stratospheric Observatory for Infrared Astronomy (SOFIA) is a Boeing 747SP carrying a 2.5 meter infrared telescope capable of operating between at altitudes of between twelve and fourteen kilometers, which is above more than 99 percent of the water vapor in the atmosphere. The ability to make observations above most water vapor coupled with the ability to make observations from anywhere, anytime, make SOFIA one of the world's premiere infrared observatories. SOFIA uses three visible light CCD imagers to assist in pointing the telescope. The data from these imagers is stored in archive files as is housekeeping data, which contains information such as boresight and area of interest locations. A tool that could both extract and process data from the archive files was developed.

Nomenclature

AOI	= area of interest
API	= application programmer interface
CCD	= charge coupled device
FFI	= fine field imager
FITS	= flexible image transport system
FPI	= focal plane imager
GUI	= graphical user interface
HK	= housekeeping
SOFIA	= Stratospheric Observatory for Infrared Astronomy
TA	= telescope assembly
TSV	= tab separated values
UI	= user interface
WFI	= wide field imager
XML	= extensible markup language

I. Introduction

SOFIA—a Boeing 747SP modified to carry a 2.5 meter infrared telescope—has three CCD imagers used for aiming the telescope. Data from the WFI, FFI, and FPI are stored in .ark files, which conform to the format defined by the SOFIA Archive Format Definition. Imager archive (.ark) files contain image serial numbers, timestamps, exposure time, and other data in addition to the image data. Housekeeping data is currently stored in a format that is processed into TSV files post flight, but it is expected that in later phases of operation the housekeeping data will be stored in a format similar to the one used by the imagers. A GUI that displays imager data with housekeeping data in real time had already been developed, but prior to the development of the SOFIA Image Processing Tool, there was no easy method for converting the archive data into standard image formats, or correlating it with housekeeping data.

The requirements for the SOFIA Image Processing Tool are that it has to be able to extract data from .ark files, overlay housekeeping data over top of the image data, perform adjustments on the images, save outputs in standard image formats, FITS format, and a standard video format. Both GUI and command line interfaces had to be provided so that it would be easy for both humans and other computer programs to use the tool. A simple method of making modifications to the tool also had to be provided to allow for: future modifications to the FITS header, planned changes in the format in which housekeeping data is saved, and additional data to be overlaid on images. Part way through development, a requirement was added for the tool to be able to display images from all three

[*] Intern, Flight Instrumentation, Dryden Flight Research Center, University of California, Santa Barbara
[†]Instrumentation Engineer, Flight Instrumentation, Dryden Flight Research Center

imagers at the same time. The program had to be written in a programming language that could be maintained by people at Dryden, and had to be cross platform. For these reasons, it was decided that the tool should be written in Java.

II. Background Information

A. Archive File Format (.ark)

A basic understanding of the ark file format is necessary for understanding the design of the archive extractor portion of the tool. Data from the WFI, FFI, and FPI are stored in the archive file format defined by the SOFIA Archive Format Definition. Files in this format consist of a XML header followed by a series of data records. The XML header contains different data record formats, each of which lists the name, representation, and order of appearance of values in the body of data records.

Each data record contains a header consisting of a sync word, an integer value telling the size of the record, a timestamp, and a null terminated string containing the format name of the data record. The format name can be used in conjunction with the XML header to parse the contents of the body of the data record. The body of the data record contains values stored sequentially in memory. Variable length values are prefixed by a four byte integer telling the size of that data entry.

B. Plugin API

It is expected that changes will be made to the SOFIA Image Processing Tool in the future. The format in which HK data is stored is expected to change soon, the format of the FITS header may change, and users may want to overlay additional information on images or save data in different formats. All of these changes could be made by modifying the source code of the tool and recompiling it, but providing a plugin API makes it far easier for people who are not already familiar with the source code to make changes and additions to a program. Plugins are generally stored in separate files which contain functions or classes. Plugins interface with the host program through some pre-specified interface. All plugins discussed in this paper are java classes stored in .jar files. A given plugin API will specify the functions that a plugin class should implement, and the functions provided by the tool to the plugin class. All communication between plugins and the tool are made through function calls.

III. Software Architecture

There have been several major changes in the software architecture since the creation of the image processing tool. The software architecture originally did not contain a plugin API and was designed to only manage archive files for one imager. An imager archive extraction plugin API has not yet been added because by the time it was decided that a plugin API would be useful, the .ark file extractor was integrated with the rest of the code in a manner that would have made it difficult to easily add a plugin API for it, and because the .ark file format is not expected to change in the future.

The software can be broken down into three main components: UI, plugins, and imager archive file extraction and organization. The user interface processes user input and then provides appropriate instructions to the .ark file organizer and any active plugins. The plugins may request data from the archive file organizer while performing their functions. For example, if the user is running the tool in GUI mode and requests to view the next frame, the program would first issue the next frame command to the .ark file organizer. The organizer would return control to the GUI when this operation was complete. Next, the GUI would request that the current display plugin convert the raw image data provided by the .ark file organizer into a displayable image. When the display plugin returns the image, the UI would then ask the display plugin to perform its overlay operation on the image. If the default display plugin was loaded, this would involve the display plugin requesting housekeeping data from the HK plugin, overlaying the data on the image, and returning the modified image. Finally, the GUI would display the image returned by the display plugin.

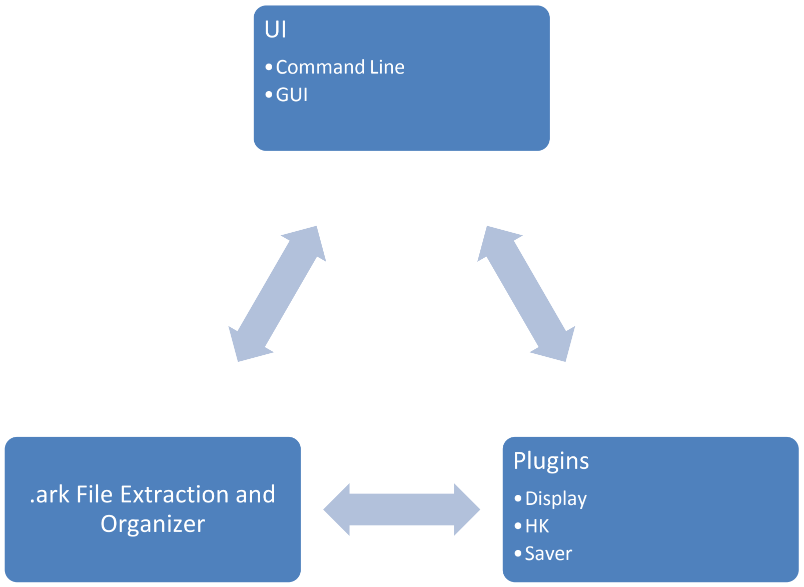


Figure 1. A high level view of the SOFIA Imager Archive Tool software architecture.

IV. The Archive File Extractor

The imager archive file extractor was developed in three stages. The first version did not sort images based on from which imager they came and was written prior to the requirement for displaying images from all three imagers simultaneously. The second version of the archive file extractor was written to allow images from all three imagers to be opened and viewed simultaneously. The second version used more memory than was available on the standard Java heap, so a third version was written that uses less memory and has faster seek times to arbitrary timestamps.

A. First Version of the Archive File Extractor

The first version of the archive file extractor loaded entire archives into memory and sorted archives by timestamp. To load an individual archive file into memory, the archive loader would first read the XML header to determine the data record formats. The loader would then read the entire file into a series of data structures constructed based on the XML header.

In the first design iteration, only one archive file was loaded into memory at a time, but this caused lag when a user was browsing the archive and changed to the next file, as changing to the next file required that the current file be removed from memory and the next file be read from the disk. Code was added to the program to start new threads that would load the previous and next files in the background, and the first and last archive files were kept in memory at all times. Doing this removed lag when going to the next or previous frame, or when jumping to the first or last frames in a group of archive files. However, this did not remove lag time when jumping forwards or backwards in time, and increased memory usage by imager archives by a factor of almost five.

B. Second Version of the Archive File Extractor

The second version of the imager archive file extractor was developed to support opening of files from all three imagers at the same time. To do this, an “archive organizer” was created. The organizer allows one imager to be designated as the “master”. The images displayed by the other imagers are then selected so that the timestamps are as close to the timestamp on the image being displayed from the master imager as possible. Memory use in this second version was unacceptably high, and all frame changing operations included one or more seconds of lag.

C. Third Version of the Archive File Extractor

The third version of the imager archive file extraction system -- a near complete rewrite of how archive files were loaded and stored in memory – was developed to fix the memory and performance problems present in the first and second versions. The memory usage was reduced by only keeping a single archive entry in memory at a time. The seek time to an arbitrary timestamp was reduced by keeping the locations of each sync word in an archive file in memory, and associating the timestamps and serial numbers with the sync words.

In the first implementation of this system, all archive files were opened and scanned for sync words and timestamps each time they were opened, which led to excessive load times. To resolve this, the first time an archive file is opened the program stores the sync word locations, serial numbers, and timestamp in text files on the disk, allowing the program to find all sync words, timestamps, and serial numbers by reading a much smaller file on subsequent opens.

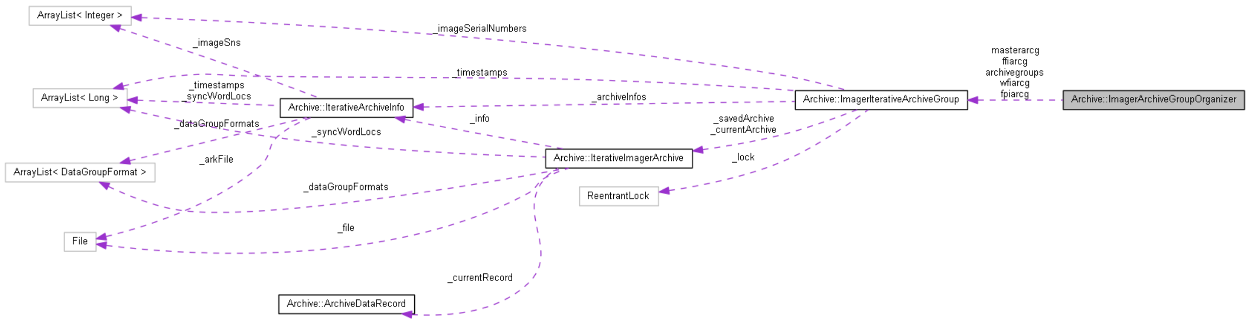


Figure 2. Collaboration diagram for the archive group organizer in the third version of the archive file extractor.

V. The House Keeping Extractor

The housekeeping data is currently stored in TSV files, but the format in which the data are stored is expected to change to something similar to what is used to store the imager archive files. To support this change, and any other future changes in the housekeeping data format, a housekeeping plugin API was developed. The Housekeeping plugin API defines two abstract classes: a housekeeping loader plugin, and a housekeeping data source. The housekeeping loader plugin class contains a function for loading housekeeping data from one or more files which returns a housekeeping data source object. The housekeeping data source provides an interface for getting housekeeping data keyed by a string and a timestamp. When trying to open housekeeping data, the tool first loads a housekeeping loader plugin class definition from a jar file. It then uses that class to create a housekeeping loader plugin object. Next, it calls that object's load method on the file containing housekeeping data, and stores the return value for later use. A HK loader plugins were developed both to support the current TSV HK file format and the planned future format.

VI. Display Plugin API

In the future, people will likely want to overlay additional information over the images, and may want to provide image adjustments not currently provided such as color maps. The display plugin API makes adding features such as these easier than it would be if modifying the source code was necessary. The plugin API allows plugins to modify the GUI, determine how raw image data is rendered and control what data is overlaid on images.

All display plugins extend (i.e. are child classes of) the DisplayPlugin class. The display plugin class has two abstract methods that child classes are required to override: imageFromImagerData and overlay. The imageFromImagerData method converts raw image data into a BufferedImage object. How this is performed is defined by the plugin. The default plugin maps the fourteen bit data to an eight bit image, and may also perform an additional linear correction or histogram equalization. The overlay method is responsible for overlaying data on top of the image. If the image is resized, the overlay operation is performed after the imageFromImagerData operation because resizing text can cause it to become unreadable and resizing one pixel wide lines can cause them to disappear. The DisplayPlugin class also contains a popup menu that is shown when the tool is running in GUI mode and a user right clicks on an image. Subclasses can customize this menu.

VII. Saver Plugin API

The saver plugin API provides an easy way to add support for saving data in new file formats. All saver plugins are child classes of the SaverPlugin class. Saver plugins have three abstract methods that sub classes must override: getName, nextImage, and done. The getName method should be overloaded to return the name of the plugin. The GUI uses getName to fill out the save type dropdown, and the command line UI uses get name to select the plugin to use based on the command line arguments. The nextImage method is called once for each image that should be saved. The done method is called after nextImage has been called once on each image to be saved.

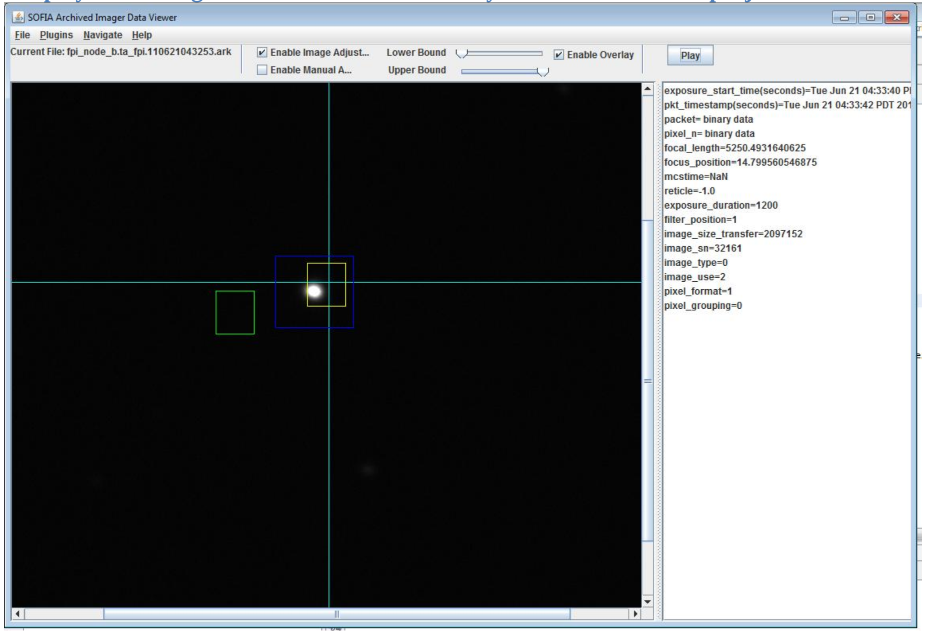
Five saver plugins are included with the default build, each of which is responsible for saving a different file type. The currently supported file types are: bitmap, GIF, JPEG, FITS, and MOV (QuickTime movie). The FITS saver plugin always saves the raw, unadjusted images. The other saver plugins save the images after using the active display plugin to perform adjustments and overlaying of housekeeping data.

VIII. User Interface

Both GUI and command line interfaces are provided by the SOFIA Image Processing Tool. The GUI is meant for use by humans, whereas the command line interface allows other computer programs to easily interface with the tool. The main program launches the GUI interface when it is run with no command line arguments and the command line interface when it is run with command line arguments.

A. GUI Interface

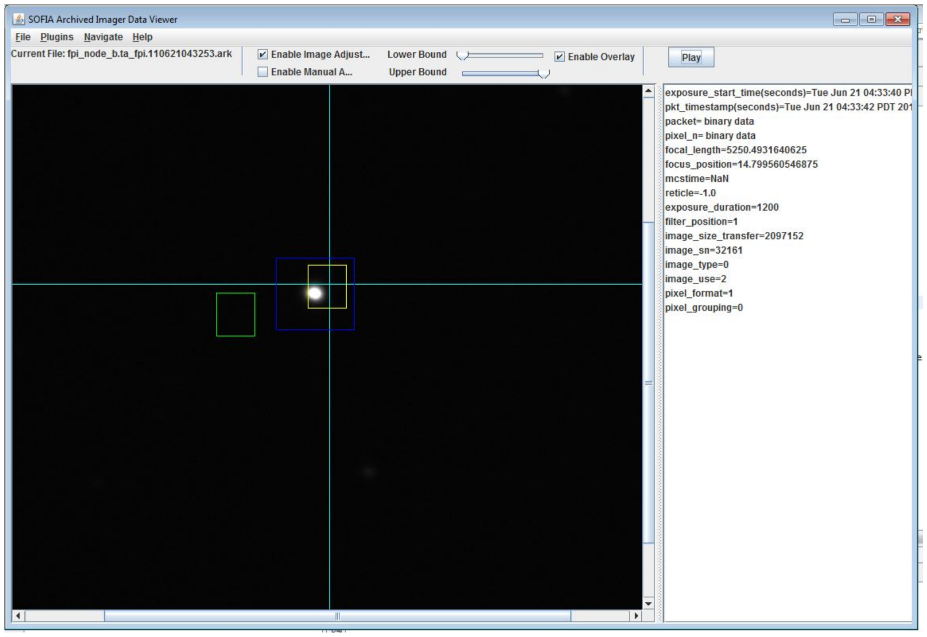
A GUI interface is provided to allow humans to more easily interact with the tool. The GUI allows users to view or save archive file content. The first GUI interface developed only displayed one image at a time because, when it was developed, the requirement to display three images at the same time had not yet been added to the project. A screen shot of this early GUI is shown in



Current File: fpi_node_b.ta_fpi.110621043253.ark

```
exposure_start_time(seconds)=Tue Jun 21 04:33:40 PI
pkt_timestamp(seconds)=Tue Jun 21 04:33:42 PDT 201
packet= binary data
pixel_n= binary data
focal_length=5250.4931640625
focus_position=14.799560546875
mcstime=NaN
reticle=-1.0
exposure_duration=1200
filter_position=1
image_size_transfer=2097152
image_sn=32161
image_type=0
image_use=2
pixel_format=1
pixel_grouping=0
```

Figure 3. This GUI displayed an image next to information about the image. The toolbar contained checkboxes and sliders for interacting with the display plugin. This GUI used the first version of the archive file extractor to get image data, and did not support housekeeping data loader plugins, or saver plugins.

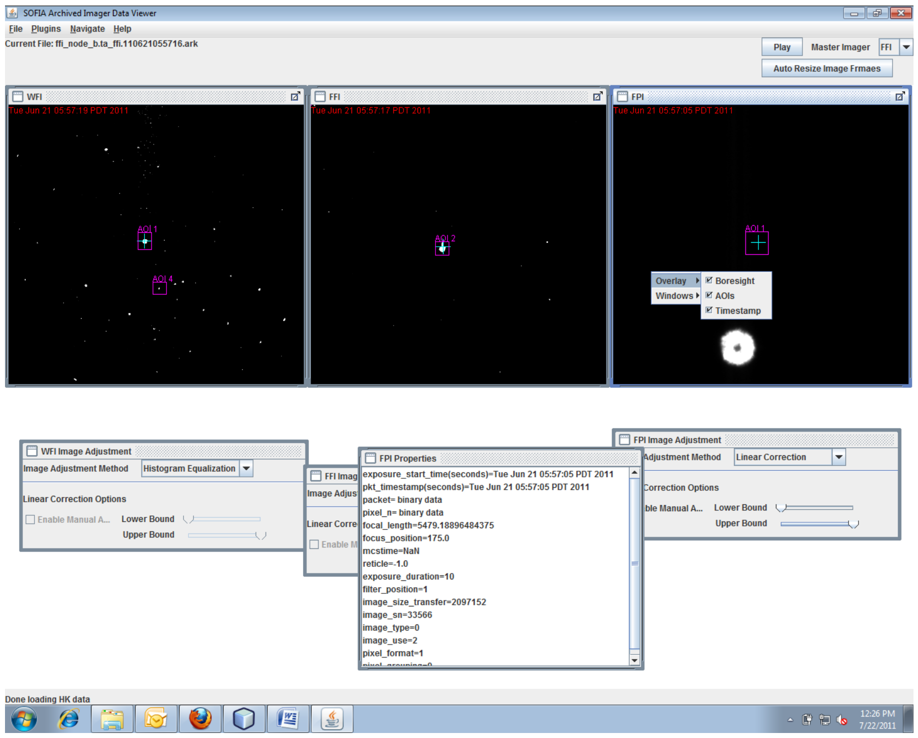


Current File: fpi_node_b.ta_fpi.110621043253.ark

```
exposure_start_time(seconds)=Tue Jun 21 04:33:40 PI
pkt_timestamp(seconds)=Tue Jun 21 04:33:42 PDT 201
packet= binary data
pixel_n= binary data
focal_length=5250.4931640625
focus_position=14.799560546875
mcstime=NaN
reticle=-1.0
exposure_duration=1200
filter_position=1
image_size_transfer=2097152
image_sn=32161
image_type=0
image_use=2
pixel_format=1
pixel_grouping=0
```

Figure 3. The first version of the GUI. The teal cross hairs show the position of the boresight. The rectangles show AOIs, with each AOI shown in a different color. The panel on the right displays all other information that was stored in the same data record as the image.

The second and final GUI interface developed was heavily a modified version of the first GUI interface, containing a desktop pane on which internal windows can be moved around and resized. All interaction with the display plugin is done through a right click menu. This version of the GUI is shown in Figure 4. The second GUI was designed to look more like the TA operator's GUI used in flight. As such, the default display plugin was changed so that AOIs are all purple and labeled by number, and the boresight overlays are smaller. Also, at startup the three imager displays are positioned identically to the TA operator's GUI.



Current File: ffi_node_b.ta_ffi.110621055716.ark

WFI: Tue Jun 21 05:57:19 PDT 2011
FFI: Tue Jun 21 05:57:17 PDT 2011
FPI: Tue Jun 21 05:57:05 PDT 2011

WFI Image Adjustment
Image Adjustment Method: Histogram Equalization
Linear Correction Options
Enable Manual A...

FPI Image Adjustment
Adjustment Method: Linear Correction
Correction Options
ble Manual A...

FPI Properties

```
exposure_start_time(seconds)=Tue Jun 21 05:57:05 PDT 2011
pkt_timestamp(seconds)=Tue Jun 21 05:57:05 PDT 2011
packet= binary data
pixel_n= binary data
focal_length=5479.18896484375
focus_position=175.0
mcstime=NaN
reticle=-1.0
exposure_duration=10
filter_position=1
image_size_transfer=2097152
image_sn=33566
image_type=0
image_use=2
pixel_format=1
```

Done loading HK data

Figure 4. The second version of the GUI.

B. Command Line Interface

The command line interface provides a way for other computer programs to interact with the SOFIA Image Processing Tool. The command line interface allows saving of archive file contents but, unlike the GUI, provides no method for browsing archive files.

IX. Conclusion

All project requirements were successfully completed. Possible future improvements to the tool include making an imager archive plugin API and adding an image colorization feature to the default display plugin. A major overhaul of the software to make it less monolithic would make future development easier.