# A Flexible Object Oriented Design for Page Formating.

Nancy RA. Wolk

*Harvard-Smithsonian Center for Astrophysics, Cambridge MA, 02138*

**Abstract.** The *Chandra* Standard Data processing now includes a group of summary pages that offer a synopsis of the observation. *Chandra's* instrument and grating combinations form many different spacecraft configurations. For each configuration, a specific summary of the observation is required. We need a flexible and expandable page formatter to handle this situation. One result of this development is the *sum_format_page* tool. This C++ tool is built on object oriented design principals and contrain the flexibility to produce multiple output file formats.Here we discuss the motivations for the tool, the design and implementation, and future enhancements that need to be considered.

## 1. Introduction

The *Chandra* X-ray Observatory provides two science instruments, a transmission grating, and multiple spacecraft configurations for observers to explore the X-ray universe. The cost of the convenience to the observers is the challenge of developing software that can process these data with minimal human intervention. *Chandra's* Standard Data Processing now creates a data product that summarizes the observation for the principal investigator. The software written for this task needs to be configurable, and fairly simple to update when there are changes in spacecraft operations or enhancement requests. One component of this software is the *sum_format_page* tool. This C++ tool is designed to work with the configurable nature of the summary package.

## 2. Design considerations

Since this tool is a portion of a larger suite, we need to look at the specifications for the entire package. The specification process for an observation summary product can be subjective. Each scientist has their own view of what data and how these data should be presented. The specification process for the summary products involves polling several scientists for their opinion of what the summary of an observation should contain. The design is based on the items that most scientists want; images, details of the instrument and observing setup, sources and a quick extracted spectrum if the observation included a grating. Finally, all tools we write need to conform to the *Chandra* X-ray Center Data System (CXCDS) standards.

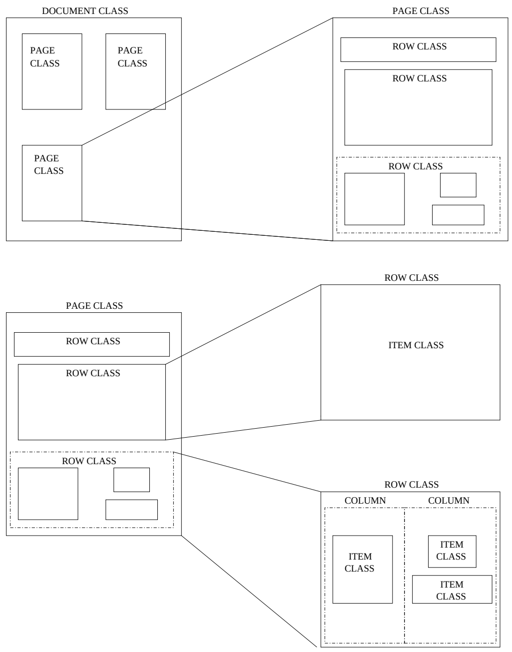


Figure 1. The relationship between the main document classes

### 2.1. Resulting Design

The result of our discussions with scientists set up the initial requirements. To control the arrangement of the items, we employ an ASCII layout template. The main document classes are designed around this template. Each row of the layout template represents an item. Items are the input information from which a document is created. An item contains information about the specific input formats and can be written in all valid output formats.

The row class contains items. We allow multiple items in a row instead of implementing a column class. Since the user may want separate pages, rows are stored in pages and the pages are stored in the document. The main function of the program reads the template then populates the document and writes the document in the requested format.

## 3. Classes

Each specific section of the design is programmed as a class. Here we discuss the basic design of the major classes. Figure 1 displays the relationships between these classes.

### 3.1. Template Reading

Two classes work to parse and store the ASCII layout template. The first class, *sumTemplateReader*, reads each line in the template, passing then to a new instance of the *sumTemplateLine*. This class parses the line and stores the item size, type, title and file information within the class. Each *sumTemplateLine* is returned and stored in standard template Vector in the *sumTemplateReader* class.

The ASCII template format allows the user to customize the document arrangement. Each line describes the item type, size, position within the row, an optional title, and the file that contains the item. The *sumTemplateReader* is passed back to the main to be used in the document class.

### 3.2. Document formation

The document is built from the *sumTemplateReader* class. The document class only contains methods to populate itself, check the physical sizes to ensure it will fit on a hardcopy, and write itself to a file. In populating the document, the layout template is read and each page in the document is filled by creating the rows and items to be used in the document.

### 3.3. Rows and Items

Each row of the document can contain multiple items which allows for columns, with the caveat that the row size cannot exceed the page size for hardcopy formats. The row is responsible for creating the individual item classes and stores the resulting objects in a vector.

Items are the specific inputs to the document. Some are predefined in the source code, such as a horizontal line. Other items are stored in files that are inserted into the document. A figure is an example of this type of item. Figures are expected to be in the correct format for the requested output. For the Latex format, this is postscript or encapsulated postscript. In the case of an HTML output, the format can be any graphic type allowed in HTML. The currently available input types are lines, new rows, new pages, figures, embedded files,links and table of contents(TOC). Embedded types are files that are in the format of the output file. These are copied verbatim from the source to the output file. The links and TOC items are only used in the HTML format to allow links to other documents and to create local links between the pages and items.

The item classes utilize the polymorphic nature of C++. The item classes are all derived from a singular base class. This class contains virtual functions for writing the outputs in the different formats. The goal of this design is to have the row iterate over its container, writing each item without having to know its type. This design has worked well so far, but recent design discussions has suggested another method of handling item classes as detailed below.

## 4. Enhancements and Upgrades

One of the design goals of the *sum_format_page* is to make enhancements and upgrades fairly simple. The input formats and file types may change in the future as well as the output formats. For these reasons, the design is flexible enough to modify the input and output formats if the requirements change.

### 4.1. Input file type changes

If a new input type or file is needed, the code can be modified to accept new formats. The changes are fairly contained and not terribly extensive. A new item class will need to be defined that is derived from the sumPageItem base class. This class will need all of the virtual functions that are defined in the base class. Once the class is written, the *sumTemplateLine* will need a new identifier in the enumeration of the item types, the ASCII template will need a new identifier to specify the type, and the row class will need to be updated with the new item class to be filled.

At the ADASS X meeting, Ben Dorman (private communication) suggested using a registry to store the input types. Any future upgrades would involve a change in the registry and overloading of functions instead of creating a new class. This is incorporacted in the redesign of XSPEC(Dorman *et al.*, 2000) and merits future exploration.

### 4.2. Output format updates

The current output formats may become obsolete in time and a new set of output formats will be needed. In this case, the tool will need to change. The changes will involve adding new functions to write the output format to each of the item classes and the base item class. The parameter file will need to be updated to accept the new format as a valid parameter. This does require the writing of new code to handle the new format for each type, but this is a necessity.

## 5. Conclusions

The summary package is a powerful data product distributed to the *Chandra* observers. It could not have been possible without the flexibility of the *sum_format_page* tool. While discussions have shown that we can make the tool even easier to upgrade with future input types, the current design is robust and can work with the multiple templates built for standard data processing.

We expect this tool, and the other tools in the summary package suite, to be used for the remainder of the mission with only minor updates. New configurations of the telescope can be handled with changes in building the ascii layout templates,whilenew data formats can be changed by updating the *sum_format_page* tool.

## 6. Acknowledgments

The author would like to thank Douglas McElroy and Kenny Glotfelty for assistance and discussions in the design phase of this project. This work was funded by the *Chandra X-Ray Center* NASA contract NAS8-39073.

## References

Dorman, B. & Arnaud, K. 2000, this volume[P1.15]