Astronomical Data Analysis Software and Systems X
ASP Conference Series, Vol. 238, 2001
F. R. Harnden Jr., F. A. Primini, and H. E. Payne, eds.

ADS's Dexter Data Extraction Applet

Markus Demleitner, Alberto Accomazzi, Günther Eichhorn, Carolyn S. Grant, Michael J. Kurtz, Stephen S. Murray

Harvard-Smithsonian Center for Astrophysics, 60 Garden Street, Cambridge, MA 02138

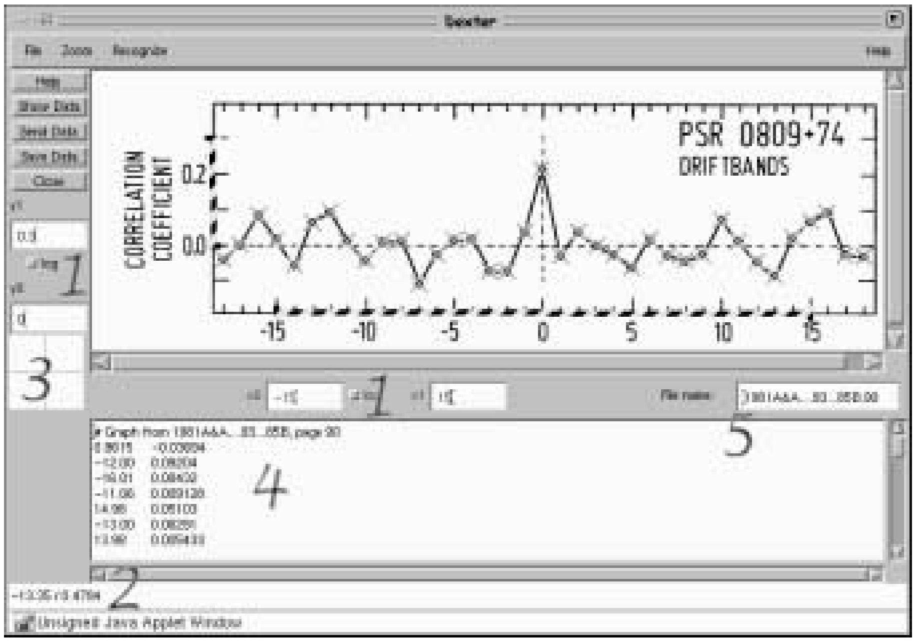
Abstract. The NASA Astrophysics Data System (ADS) now holds 1.3 million scanned pages, containing numerous plots and figures for which the original data sets are lost or inaccessible. The availability of scans of the figures can significantly ease the regeneration of the data sets. For this purpose, the ADS has developed Dexter, a Java applet that supports the user in this process. Dexter's basic functionality is to let the user manually digitize a plot by marking points and defining the coordinate transformation from the logical to the physical coordinate system. Advanced features include automatic identification of axes, tracing lines and finding points matching a template. This contribution both describes the operation of Dexter from a user's point of view and discusses some of the architectural issues we faced during implementation.

1. Dexter's Operation

The ADS provides access to the full-text of over 200,000 scientific papers published in astronomical journals, conference proceedings, newsletters, bulletins and books, for a total of over 1.3 million pages. The ADS article service allows users to view individual pages using any web browser with graphical capabilities. When viewing a scanned page, the Dexter applet can be started by following the link available below the image. As described in its help page[1], Dexter can easily be used to extract data; a relatively simple case is depicted in Figure 1.

After starting Dexter, the user can select the portion of the page containing a plot or figure to be analyzed. (It is advisable to keep this portion as small as possible, both to reduce the Java Virtual machine memory requirements and to facilitate the automatic feature extraction algorithms.) When Dexter's main window has popped up, it is generally worthwhile to attempt an automatic detection of the axes (in the Recognize menu). In the example of Fig. 1, this has worked well; in other cases it may be necessary to mark the axes manually or correct Dexter's axes by click-and-dragging the ends of the axes. Next one fills in the text fields for the start and end values of the axes (marked with a large "1" in Fig. 1), which completes the information needed by the applet to display

[1]`http://adsabs.harvard.edu/Dexter/Dexterhelp.html`



```
# Graph from 1981A&A...93..85B, page 30
0.8615    -0.03694
-12.00    0.08204
-16.01    0.08409
-11.06    0.089128
14.98     0.05103
-13.00    0.08291
13.98     0.089433
```

```
-13.35 r0.4764
```

Figure 1. An illustration of Dexter's main window after the data from a graph was extracted. Marked points and axis markings have been emphasized to enhance their visibility in this grey-scale rendering.

physical (graph) instead of logical (pixel) coordinates in the status line ("2" in Fig. 1).

To actually mark points (the crosses in the figure), one uses the "Find Points" function from the Recognize menu and clicks on a sample point. Dexter then marks all similar points, where the similarity threshold can be adjusted in "Recognizer Settings". Occasionally Dexter will miss overlapping points and encounter similar mishaps, or it may have a hard time following the correct path when tracing lines. In these cases, it is necessary to mark the points manually, which is done by clicking on their positions in the graph. Error bars can be added by clicking on a point and dragging the bar (error bars are only available after the coordinate system has been set up). The magnifying glass ("3" in Fig. 1) may help here and can be activated by clicking on it. To work around bugs in some Java virtual machines, it is inactive by default.

When all the points are marked, any method listed in the File menu can be used to obtain the resulting data set – "Show Data" outputs it in the text field labeled "4" in Fig. 1, "Send Data" usually opens a browser window, and "Save Data" will save it in a text file on the user's machine (provided the browser is correctly configured). The name of the text file can be set in the "File name:" field ("5" in Fig. 1) and defaults to "bibcode.page" (bibcode being the article's bibliographic code and page the page's sequential number within the article).

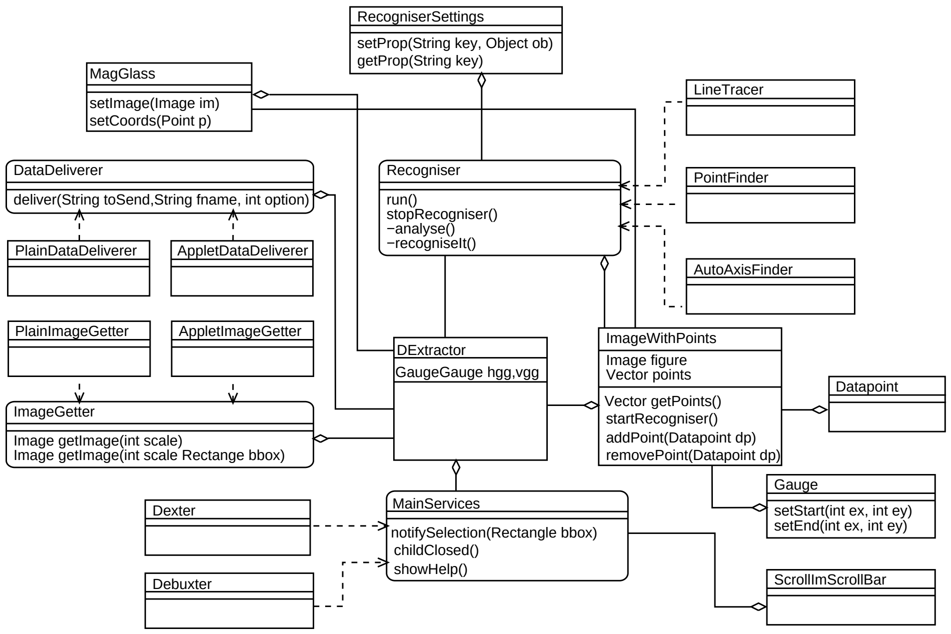


Figure 2. An abridged representation of Dexter's architecture.

2. Dexter's Architecture

Fig. 2 shows a raw sketch of Dexter's architecture in a graph inspired by the Unified Modeling Language (UML). Rectangles symbolise classes, rounded rectangles stand for interfaces, with dashed arrows from their implementation. Connecting lines indicate that two classes are talking to each other, with the diamond pointing from an embedding class.

In the central position there is the DExtractor class that implements most of the user interface and contains the logic responsible for the transformation from screen to graph coordinates through the GaugeGauge class that in turn controls the text fields for entering the start and end values for the graphical gauges. The DExtractor class is derived from the Abstract Windowing Toolkit's Frame class.

The panel containing the image of the figure is displayed by a different class, ImageWithPoints, that also embeds the classes handling the graphical Gauges and the Datapoints. These latter classes handle their events themselves. For example, the handling of error bars is completely encapsulated within the Datapoint class.

The MainServices interface hides the startup from DExtractor. Two implementations of MainServices exist: Dexter, the applet interface, and the standalone Debuxter useful for debugging. It should not be difficult, however, to implement MainServices that, for example, would use ghostscript to provide Dexter's capabilities for PostScript articles. To build such a PostScript-Dexter, one would also need to adjust the implementations of the ImageGetter and DataDeliverer interface, the first accepting requests for scaled and cropped versions of the image, the second delivering the extracted data after the user has requested to save the data or get it sent. Among the implementations of these interfaces, the AppletDataDeliverer is somewhat involved since an (unsigned) applet cannot access the client's disk or portably send data directly to the browser due to Java's security model. In order to save data or display it in a browser window, it is relayed back to the host and tunneled to the client through a pipe on the host.

The Recogniser interface, derived from Java's Thread class, defines how classes that try to do automatic feature extraction interact with both DExtractor and ImageWithPoints. It also contains some utility functions, for example to acquire the pixels from the graph image. Images are stored as arrays of signed bytes by the Recognisers; the dynamic range from 0 to 127 is more than adequate for the images Dexter deals with, and the sign is rather useful for flagging purposes, e.g., in the flood filler used in the PointFinder Recogniser.

Recognisers need to communicate with ImageWithPoints to access the image and to set points or gauges they may have found, as well as with DExtractor, giving prompts in the status line and telling it when they are finished. DExtractor needs to know this to re-enable some critical operations that are disabled while a Recogniser is running (changing the resolution of the graph image, sending data). Most Recognisers will need some sort of user input, e.g., to get a start point for line tracing or a template for point matching. Since this is done under control of the Recogniser thread, (almost) all that has to be done from Dexter's main thread is to make a call to the Recogniser's start Method.

Since Recognisers do not register themselves automatically with DExtractor, some source code changes in both DExtractor (that controls the menu bar in which the Recognisers are registered) and ImageWithPoints (that actually starts Recognisers) are necessary when a new Recogniser is written. Given the current scope of the project (about 5000 lines of source code), a more elaborate plug-in scheme did not seem necessary. If Recognisers have adjustable parameters, they can use the RecogniserSettings class, containing both a Hashtable to store the property values and the logic to display a dialog to change them.

All three Recognisers currently implemented (AutoAxisFinder to locate the axes, PointFinder for point matching, LineTracer for automatically digitising lines) use naive syntactic algorithms. In a tool like Dexter intended to be interactive, the computational effort for performing Fourier or Hough transforms on entire images makes these approaches currently unattractive, given the poor performance of the Java virtual machines on some architectures.

3. Conclusions

Dexter has shown itself to be a very useful tool for those interested in obtaining numerical data from ADS articles. The applet could be easily adapted by other data archives providing scanned publications, and even extended to work with postscript or PDF files. Dexter's source code is available under the GNU General Public License on Sourceforge.net[2]

[2]`http://Dexter.sourceforge.net`