GUIDE: A Sherpa Package for Analyzing High-Resolution Spectra

Chandra X-ray Center
Smithsonian Astrophysical Observatory
60 Garden St.
Cambridge, MA 02138
USA

September 28, 2001

Chapter 1

Introduction: GUIDE

G rating

U ser

I nteractive

D ata

E xtension

Guide is meant to extend the power of *Sherpa* for doing spectroscopic analysis. *Guide* has been designed to be useful at a number of different levels, depending on the user's need. At its most basic level, *Guide* enables users to easily store multiple models and different analysis sessions electronically, and then restore this information. While a user will likely still use a written log of some type, with *Guide* he/she should be able to store and retrieve most of the numeric data and inputs to *Sherpa*, rather than having to retype them. In addition, *Guide* will write this information in a FITS file known as an Model Descriptor List (MDL) file (see Chapter 2), that can be easily interpreted using prism, the CIAO FITS viewer.

At the next level, *Guide* can be used to identify individual emission features; ie, associating a gaussian model component in *Sherpa* (with a wavelength and flux determined by fitting to the data) with a Si XIII emission line. *Guide* can access collisional plasma models, such as the APEC (Smith & Brickhouse 2000) or (in the future) the Raymond & Smith (1977; 1993 update) models to get predicted line emissivities as functions of temperature and density. After the user decides upon the line identifications, *Guide* can then use those lines' model emissivities as the basis for fitting an emission measure distribution model to the fluxes. The fluxes are determined from the gaussian (or other model) fits to the line profile.

At the highest level, *Guide* provides the user with interactive programming capability. *Guide* derives much of its power from the inclusion of the *S-lang* scripting language in *Sherpa* and *CHiPs*. *S-lang* is described in great detail at `http://space.mit.edu/~davis/slang1.4-doc.html`. Much of *Guide* is implemented in *S-lang*, which has similarities to both C and IDL. It is relatively easy to extend *Guide* (and, more generally, *Sherpa*) using *S-lang* scripts; the *Guide* scripts (which can be found in the directory $ASCDS_INSTALL/bin/interpreted) can be used as examples.

1.1 *Guide*: A Quick Walkthrough

We begin with a quick walkthrough of a *Guide* session, using a Chandra HETG observation of Capella (ObsID 1318) as a sample dataset. We will begin with a type II pha file created using the standard CIAO pipeline and the calculated effective area for each grating and order ("arf" files).

Since *Guide* is an extension to *Sherpa*, we begin by starting sherpa and then loading the *Guide* extension:

```
unix% sherpa
-----------------------------------------------------
```

```
Welcome to Sherpa: CXC's Modeling and Fitting Program
-----------------------------------------------------
Version: 2.0 (2 June 2000)
sherpa> import("guide")
GUIDE Initialized
WARNING: GUIDE is BETA software.
WARNING: USE AT YOUR OWN RISK.
sherpa>
```

At this point all the *S-lang* scripts and variables which are necessary for *Guide* have been loaded. Next, we need to load the data files into *Sherpa*; in this case, although the pha datafile has 12 separate spectra, we will focus only on the ± 1 orders of the HEG and MEG gratings:

```
sherpa> data acisf01318N002_pha2.fits
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
The inferred file type is PHA Type II.  If this is not what you want, please
specify the type explicitly in the data command.
Failed to find SYS_ERR column
sherpa> paramprompt off        # More convenient in scripts
sherpa> set plot noerrorbars   # Improves appearance of plots
sherpa> rsp[hegm1]
sherpa> rsp[hegp1]
sherpa> rsp[megm1]
sherpa> rsp[megp1]
sherpa> hegm1.arf = heg_m1.arf
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> hegp1.arf = heg_p1.arf
...
sherpa> megm1.arf = meg_m1.arf
...
sherpa> megp1.arf = meg_p1.arf
...
sherpa> instrument 3 = hegm1
sherpa> instrument 4 = hegp1
sherpa> instrument 9 = megm1
sherpa> instrument 10= megp1
```

We can now begin with the usual single-temperature global fit to the spectrum. As Capella is a bright source, the lines can be easily seen in any

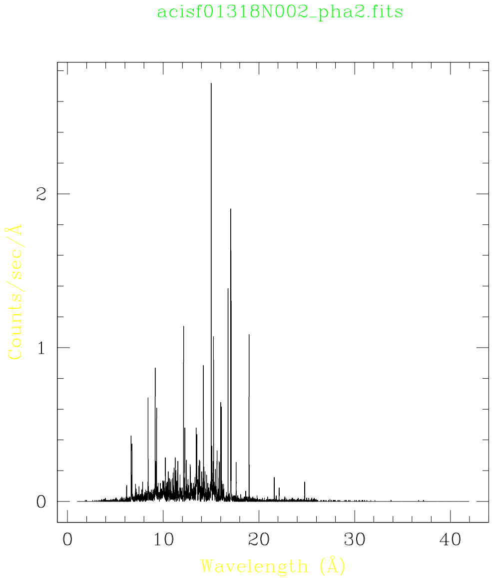

Figure 1.1: MEG -1 order spectrum of Capella, from ObsID 1318.

of the 1st order spectra, and background subtraction can be ignored. For example, dataset #9 is the MEG -1 order, and executing the command `lp data 9` should show a plot similar to Figure 1.1.

> Note that no redistribution matrix has been set (aka "rmf" files). Although generic grating rmf files have been created by the CXC, there is currently no general way for users to create them for a specific observation. When a type II pha file is loaded, *Sherpa* by default begins in wavelength mode; *i.e.*, plots are done in wavelength not energy and model fitting is also in wavelength units. However, the only models that can be fit in wavelength units currently are the gaussian, normalized gaussian, and lorentzian models. Any other models, specifically all the XSPEC models in *Sherpa* can only be fit in energy space, due to limitations in how the XSPEC models are integrated into *Sherpa* as of CIAO v2. This can be done using the `analysis # energy` command in *Sherpa*.

One way to begin is to define a simple one-temperature plasma model (here, the xsapec model) and fit it:

```
sherpa> analysis 9 energy #
sherpa> source 9 = xsapec[a1]
sherpa> fit 9
 powll: v1.2
```

```
 powll:   initial function value =     1.05432E+08
 powll:     converged to minimum =     1.29680E+04 at iteration =      3
 powll:   final function value    =     1.29680E+04
            a1.kT  0.509069  keV
            a1.norm  0.00765804  10**-14 / (4 pi D**2) Int n_e n_H dV
```

This shows the best single-temperature fit for Capella is ~ 0.509 keV or 5.9×10^6 K. This is expected—Brickhouse et al. (2000, ApJ, 530, 387) found a strong peak in the emission measure distribution around 6×10^6 K.

We now focus on individual emission lines in the data. This is better done in wavelength mode, so we will switch: `analysis 9 wave`. Figure 1.1 shows many strong lines; we will focus on those near 8.5Å. We can begin by plotting the region near 8.5Å, and then fitting a normalized gaussian to the line:

```
sherpa> ignore allsets all
sherpa> notice allsets wave 8.2:8.6
sherpa> source 3,4 = ngauss[hg1]
sherpa> source 9,10 = ngauss[mg1]
sherpa> mg1.fwhm => 2*hg1.fwhm
sherpa> mg1.ampl => hg1.ampl
sherpa> fit 3,4,9,10
 powll: v1.2
 powll:   initial function value =     3.93384E+02
 powll:     converged to minimum =     3.93215E+02 at iteration =      4
 powll:   final function value    =     3.93215E+02
           hg1.fwhm  0.0118113
           hg1.pos  8.41978
           hg1.ampl  0.000176782
           mg1.pos  8.41475
```

The results can be seen in Figure 1.2, for all 4 datasets. The *Guide* identify command gives a list of possible matches for the line :

```
sherpa> identify(8.415)
Wavelength (A)    Ion    UpperLev LowerLev Emissivity (ph cm^3/s)
   8.4193 --     Mg XII      4 ->      1,  4.770e-17 @ logT =  6.90
   8.4247 --     Mg XII      3 ->      1,  2.266e-17 @ logT =  6.90
   8.4060 --     Mg XII   1012 ->     17,  1.288e-18 @ logT =  6.90
   8.4209 --    Fe XXII    177 ->      8,  1.283e-18 @ logT =  7.10
   8.4160 --     Mg XII   1055 ->     13,  5.761e-19 @ logT =  6.90
```

```
   8.4181 --     Mg XII   1022 ->      9,  4.523e-19 @ logT =  6.90
   8.4053 --     Mg XII   1012 ->     15,  1.825e-19 @ logT =  6.90
   8.4144 --    Fe XXII    157 ->      6,  8.474e-20 @ logT =  7.10
   8.4084 --    Fe XXII    158 ->      7,  1.427e-20 @ logT =  7.10
   8.4157 --     Mg XII   1014 ->     15,  1.083e-20 @ logT =  6.90
sherpa> flux 3
Flux = 0.000176782 photons/cm**2/s
Flux = 4.17198e-13 ergs/cm**2/s
```

This table shows that the strongest lines within 0.01Å of 8.415Å are the $3 \rightarrow 1$ and $4 \rightarrow 1$ transitions of Mg XII. These are in fact the $n = 2 \rightarrow 1$ hydrogenic transitions, which can be determined using the `describe` command. The `describe` command takes as inputs the nuclear charge Z and the ionization state (expressed as the value of the roman numeral), and the upper and lower levels of the transition. Therefore, Mg XII $3 \rightarrow 1$ is `describe(12,12,3,1)` (or, alternatively, `describe(12,12,1,3)`):

```
sherpa> describe(12,12,3,1)
Ion Mg XII, energy level 1 ---
electron configuration      : 1s~^2S_{1/2}
energy above ground (eV)    : 0.000000
Quantum state               : n=1, l=N/A, s=2, degeneracy=2
Energy level data source    : 1983ADNDT..29..467S
Photoionization data source : 1964ApJS....9..185B
-------------------------------------------
Ion Mg XII, energy level 3 ---
electron configuration      : 2p~^2P_{1/2}
energy above ground (eV)    : 1469.430054
Quantum state               : n=2, l=1, s=2, degeneracy=2
Energy level data source    : 1983ADNDT..29..467S
Photoionization data source : 1964ApJS....9..185B
-------------------------------------------
Ion Mg XII, 1 - 3 interactions ---
Electron collision rate from    1 ->    3  : nonzero.
Reference bibcode                          : 1983ADNDT..29..467S
Wavelength (lab/observed) (Angstrom)       : 8.424620 +/- 0.000040
Wavelength (theory) (Angstrom)             : 8.438500
Transition rate/Einstein A (s^-1)          : 1.276330e+13
Wavelength (lab/observed) reference        : 1977JPCRD...6...3E
Wavelength (theory) reference              : 1983ADNDT..29..467S
Transition rate reference                  : 1987JPhB...20.6457F
```

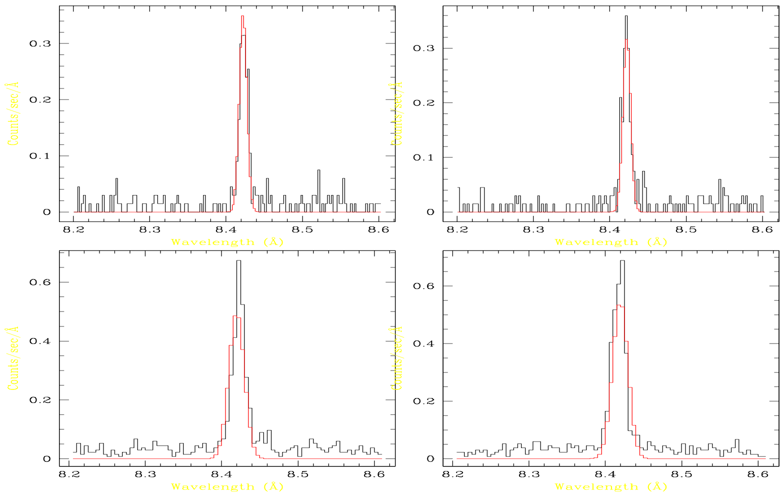

Figure 1.2: Best-fit gaussian for the line at 8.4Å, using the HETG ±1 orders of the HEG and MEG.

The exact text displayed will depend on the version of APED and GUIDE that is in use (this is from the CIAO 2.2 and APED v1.1.0.). The references are stored as ADS bibcodes (see `http://adsdoc.harvard.edu/abs_doc/bib_help.html`).

Using the *Sherpa* **`flux`** command, we get the flux for this line, albeit without any background or continuum subtraction. If we believe that this line is in fact the Mg XII, we can specifically tag it using the *Sherpa* lineid command:

```
sherpa> hg1 lineid "APECline(12,12,1,3) + APECline(12,12,1,4)"
sherpa> mg1 lineid "APECline(12,12,1,3) + APECline(12,12,1,4)"
```

The lineid command takes one of two different types of model: `APECline` or `APECrange`. The `APECline` model identifies a single emission line, such as the $3 \rightarrow 1$ (or $4 \rightarrow 1$) transition of Mg XII. Two (or more) lines can be summed (as shown above), if the observed emission line is likely a blend of two close lines. Alternatively, the `APECrange` command takes only two parameters which give an upper and lower wavelength bound on the emission lines to include. For example, `APECrange(8.41929, 8.41571)` would in-

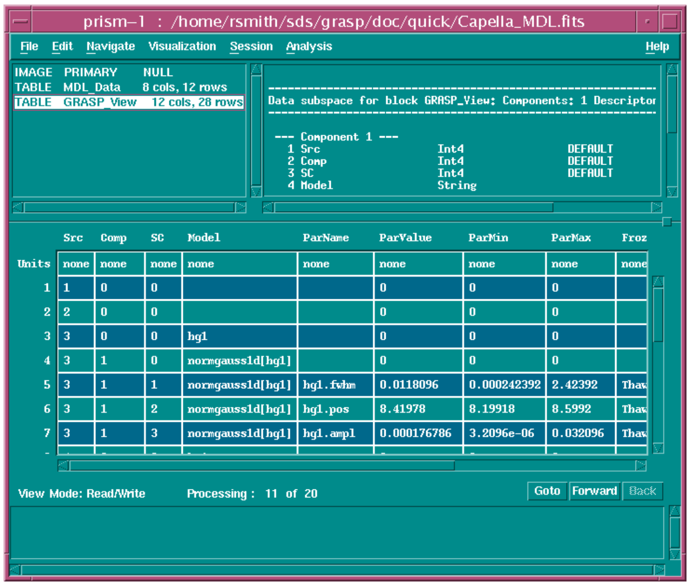

Figure 1.3: Model Descriptor List for Capella, showing the fit data for the 8.4Å Mg XII line.

clude the emissivity from all of the lines identified with the `identify(8.415)` used above.

All these results can be saved in an MDL file using the command `write MDL "Capella_MDL.fits"`, which can then be viewed in prism, as shown in Figure 1.3. See Chapter 2 for more information on MDL files.

Chapter 2

GUIDE Concepts: The Model Descriptor List (MDL)

2.1 *Guide*: The Model Descriptor List (MDL)

Since much of the point of *Guide* is the logging of intermediate and final results, the visual appearance is quite important. The "log file" is based on the Model Descriptor List (MDL) described below, but the user may choose which types of data she wishes to see at any given time. The MDL is a FITS file that stores the results a user has created during an analysis session. This means the following information must be stored:

- Data files used
- Instrument responses
- Background data
- Filters used
- *Sherpa* "sources" (Models for individual regions of the data)
- Line/Feature Identifications

Using only an MDL file and a few indices saying *which* data and source you wish to use, a user can easily restore her previous analysis session, including not only the last source used, but any source.

Sherpa can write an MDL file based on the current data and models using the command `write mdl filename.fits`. Equivalently, the command `read mdl filename.fits` will restore the previous analysis session. In addition, the contents of the MDL file are also available in the form of a *S-lang* structure by calling the *Sherpa* function sherpa_update_mdl or sherpa_get_mdl (see Chapter 3). The resulting structure can be examined using the *S-lang* "print" command:

```
unix% sherpa
sherpa> data acisf01318N002_pha2.fits
sherpa> mdlstruct = sherpa_update_mdl
_filename        =  aaaHHAK9i_
_path            =  /var/tmp/
_filter          =  NULL
_header          =  NULL
_ncols           =  20
_nrows           =  0
index            =  Integer_Type[12]
```

```
data             =  String_Type[12]
back             =  String_Type[12]
error            =  String_Type[12]
weights          =  String_Type[12]
rr_name          =  String_Type[12]
rmf              =  String_Type[12]
arf              =  String_Type[12]
src              =  Integer_Type[12]
comp             =  Integer_Type[12]
sc               =  Integer_Type[12]
model            =  String_Type[12]
parname          =  String_Type[12]
parvalue         =  Float_Type[12]
parmin           =  Float_Type[12]
parmax           =  Float_Type[12]
frozen           =  String_Type[12]
method           =  String_Type[12]
statname         =  String_Type[12]
statval          =  Float_Type[12]
sherpa> printarr(mdlstruct.data,12)
acisf01318N002_pha2.fits
acisf01318N002_pha2.fits
acisf01318N002_pha2.fits
acisf01318N002_pha2.fits
acisf01318N002_pha2.fits
acisf01318N002_pha2.fits
acisf01318N002_pha2.fits
acisf01318N002_pha2.fits
acisf01318N002_pha2.fits
acisf01318N002_pha2.fits
acisf01318N002_pha2.fits
acisf01318N002_pha2.fits
```

2.1.1 MDL: Instrumental data storage

The instrumental data to be fit must be stored. Such data include count rates, backgrounds, response matrices and so on. This does not need to be high-resolution data; any legal *Sherpa* data can be stored. For example, the following *Sherpa* commands create the MDL file shown in Figure 2.1

```
sherpa> data 1 rosatdata.pha
```

```
sherpa> back 1 rosatback.pha
sherpa> subtract 1
sherpa> RSP[pspc]
sherpa> pspc.rmf parameter value [] pspc.rmf
sherpa> pspc.arf parameter value [] pspc.arf
sherpa> pspc.eearf parameter value []
sherpa> INSTRUMENT 1 = pspc
sherpa> data 2 ascadata.pha
sherpa> back 2 ascaback.pha
sherpa> subtract 2
sherpa> RSP[gis]
sherpa> gis.rmf parameter value [] gis.rmf
sherpa> gis.arf parameter value [] gis.arf
sherpa> gis.eearf parameter value []
sherpa> INSTRUMENT 2 = gis
sherpa> data 3 einsteindata.pha
sherpa> back 3 einsteinback.pha
sherpa> subtract 3
sherpa> update
sherpa> write MDL "myMDL.fits"
```

Since *Sherpa* is a very general piece of software, the data files used can be FITS, ASCII, or IRAF IMH. The FITS and IMH columns used can be defined by the user at will, and individual columns can be read from ASCII files. Basically, data files in sherpa can either be simple, or can get very complex. The MDL data block simply stores the relevant sherpa filenames, along with an index #. Then, when *Sherpa* loads the MDL, it knows that dataset n can be loaded into sherpa by running each of the relevant commands for each column in index n.

In some cases, an RMF may not be needed; for example, if the data is from the HETG RMFs are not used, and the type II pha has the wavelength bins in it by default. In this case, the RMF column contains a NULL. In addition, for type II pha files the order is stored in this file. A type II pha file may contain multiple orders, which may be simultaneously fit but should be treated individually in the analysis.

2.1.2 Source and Model Information

One problem we have is terminology–specifically, the word "model" has a number of different meanings depending upon context. Without installing

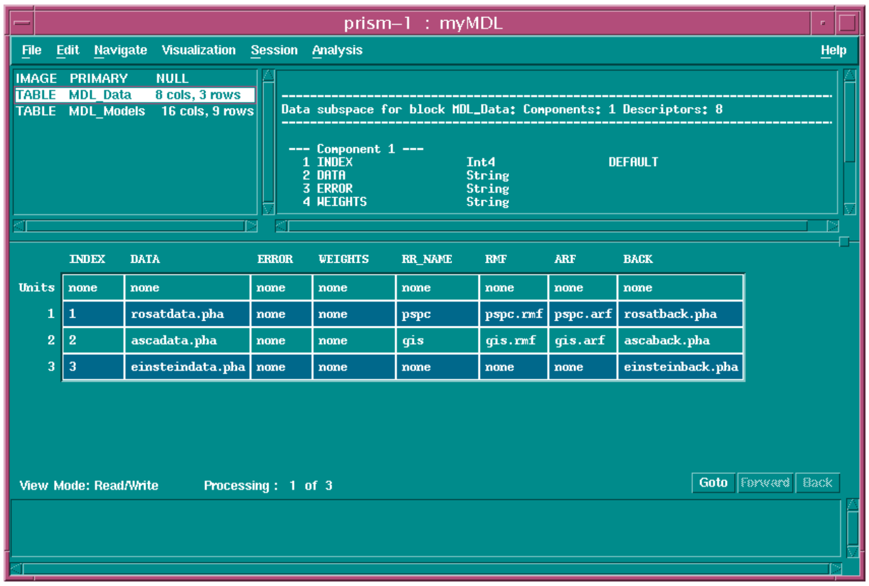

Figure 2.1: Sample Data block of a simple MDL file.

Guide, *Sherpa* has both "sources" and "models;" a source is simply a model (or combination of models) which is being used to fit the observed spectrum found in a dataset. For example, the *Sherpa* command `source 9 = xswabs[w1]*xsapec[a1]` defines a source for data 9 that consists of a combination of two models, in this case an absorption model (w1) and a collisionally-ionized plasma model (a1). This could have been broken into multiple commands, first defining the individual model elements with `xswabs[w1] xsapec[a1]` and then defining the new source with `source 9 = w1*a1`. When *Guide* is installed into *Sherpa*, the line identifications which become available are similar to *Sherpa* models. A line identification in *Guide* tells *Sherpa* to search an emissivity file to get the emissivity of a particular spectral line as a function of temperature and/or density. However, line identifications are applied to models (a particular gaussian model might be identified as a single line, or a bunch of lines), while models are applied to observed data directly.

All of this information about sources and line identifications is stored in the second section of the MDL. Exactly how this is done is complicated, due to the generality of sherpa. A model might be as simple as:

```
sherpa> source = gauss[g1]
```

or it might easily be as complex as this case, where the source is a simple gaussian but its two parameter values are also models to be fit (this is a so-called “nested model”):

```
sherpa> SRC = GAUSS[modelb]
sherpa> PositionVariation = POLY[modela]
sherpa> fwhmVariation = POLY[modelaa]
sherpa> modelb.pos => PositionVariation
sherpa> modelb.fwhm => fwhmVariation
```

In a sherpa model, we have the following possible items for each parameter in the model:

- Parameter name
- Parameter value
- Error
- Upper limit on parameter value
- Lower limit on parameter value
- If parameter is frozen in fit
- The fit method (Powell, Grid, etc)
- Parameter value is linked to other parameter values
- Parameter value is a nested model

A simple model block from the Capella MDL is shown in Figure 2.2.

The “Src” column refers to the instrumental portion of the MDL, and stores the data index for the source (*i.e.*, source # ...). Each source will have a least a few rows, and some will have many rows of information in the case of complex sources. The component index separates the source information into individual model components. Component index 0 will always have the sherpa source string, in sherpa syntax in the column “Model Defn”: e.g, `w1*g1` or `w1*(r1 + w2*r2) + g1`. In these examples, `w1`, `g1`, `r1`, and so on are sherpa model names which will be defined in the rows to follow (again in the column “Model Defn”), each with its own “component index.” So for source index 3, component 1, the column “Model Defn” is `xsabs[w1]`. The subcomponent index (SC) breaks the individual model components into their parameters. So source index 3, component index 1, subcomponent index 1, the Model column is still `xsabs[w1]` and also the ParName column is `w1.nH`.

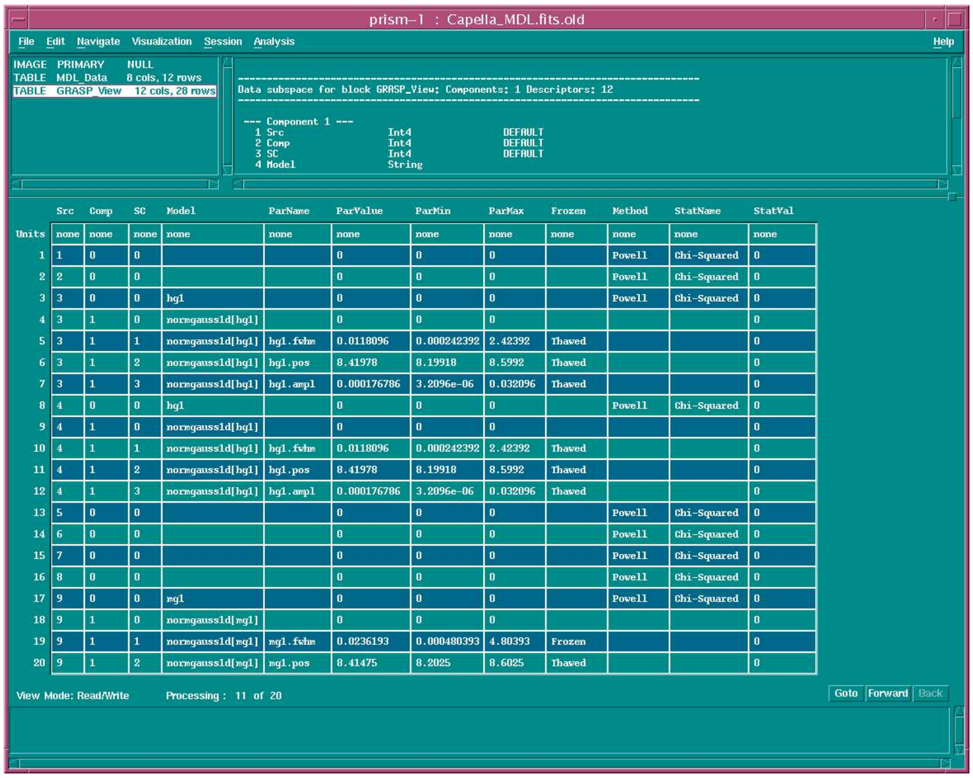

Figure 2.2: A simple model block showing a *Sherpa* session with 2 models, hg1 and mg1 and the associated data.

Then the ParValue column gives its value, the error column the fit error, and Fit Method the type of fitting (Powell, Grid, Levenberg-Marquardt, etc). The ParMin and ParMax columns set the minimum and maximum fit values, and the Frozen column notes if the parameter is frozen or not.

The Label column will hold a user-defined string of information, basically a comment on this particular model component. Finally, the LineID column will hold a *Guide* line identification, identifying the particular model component with a particular plasma feature, such as an emission line. This will be discussed in greater detail later.

2.2 Known Issues with MDL Files

MDL files are one of the most recent additions to Sherpa, and there remain a number of issues/bugs with them. While not comprehensive, this list itemizes the problems known at release and workarounds, if available.

1. When reloading an MDL file with the `read mdl mdl_file.fits` command, *Sherpa* prints a number of warning and error messages. In every tested case, however, the file is read and interpreted correctly.

2. Only models which are currently being used as sources for at least one dataset are stored in the MDL file. For example, if two separate emission lines (e.g., models g1 and g2) are fitted in sequence (`source 1 = g1` and `fit`, followed by `source 1 = g2` and `fit`, the g1 model will not be written to the MDL file unless the source is redefined as `source 1 = g1 + g2`. The models do not need to be refit.

3. When fitting emission lines (using gauss, ngauss, lorentz, or delta models), MDL files should store the flux in the line and fitted error on that flux. This flux and error is then used when fitting differential emission measure models. However, this flux and error is zeroed if a subsequent fit is done which does not include that model. This is, admittedly, a serious bug which we hope to have fixed in CIAO 2.1. The only current workaround in CIAO is to fit all the emission lines and backgrounds in one single fit. This can be done (including backgrounds) using multiple emission lines with multiple "box" models to simulate a flat background around the emission line. An alternative workaround would be to write a separate MDL file after fitting each line, and then extract the fluxes and errors for each line and add them to the final MDL file. Although we do not yet have tools to work with MDL files directly, this can be done with the FTOOL "fv."

Chapter 3

GUIDE Command Reference

Guide commands come in two flavors: pure *Sherpa* commands, and commands which actually call *S-lang* scripts. This results in a moderate amount of mixed conventions, which is unfortunate but at this point unavoidable.

3.0.1 Currently Available *Sherpa* Commands Useful for *Guide*

write mdl "filename.fits" : Writes the current data to an existing MDL logfile, or creates a new one.

read mdl "filename.fits" : Reads an already-created MDL file and restores *Sherpa* to the state given in the file.

modelname lineid "lineIDstring" : Defines a *Guide* lineID, which is similar to a sherpa source. There are two legal types, "APECline" and "APECrange". Each identifies a specific line or lines from the APEC models, and fixes a specific emissitivy as a function of temperature for the flux in this model. APECline takes four parameters– *i.e.*, APECline(atomic number, ionization stage, lowerlevel, upperlevel); the ionization stage is 1 for neutral atoms and is equal to the atomic number for the hydrogenic state. The lowerlevel and upperlevel values are the lower level and upper level of the transition, as given by the "identify" command. APECrange takes two parameters, *i.e.*, APECrange(lowerlambda, upperlambda), and includes all the lines between (but not including) these values.

sherpa_update_mdl : This routine returns an *S-lang* structure which contains all of the data which would be stored by *Sherpa* in an MDL file (e.g., data file names, source model names, model parameter values, etc.) and also forces certain calculations to be done (e.g., if there are line models present, then fluxes for those lines are recalculated).

sherpa_get_mdl : This routine returns the same data as sherpa_update_mdl, but does not force fluxes to be recalculated. If you want to be absolutely sure that the fluxes are right, use sherpa_update_mdl; otherwise sherpa_get_mdl is just as good and a bit faster.

sherpa_get_mdl_data : This routine returns only the data portion of the MDL (i.e., the list of data file names, error file names, and RMF and ARF files if any). No source model information is returned, and as result the routine is somewhat faster.

sherpa_get_mdl_models : Complementing sherpa_get_mdl_data, this routine returns only the model block (i.e., the list of source models, with values and ranges for the various model parameters). If there are no source models defined, and the user calls either sherpa_update_mdl or sherpa_get_mdl, then these functions act like sherpa_get_mdl_data and return only the data block.

3.0.2 Currently Available *S-lang* Commands

The following five commands are documented in the ahelp system and are the primary GUIDE routines for spectroscopic analysis:

identify : *identify(lambda,[dlambda, minEmissivity])* : Given a wavelength, and (optionally) a range and minimum emissivity, identify prints out a list of all strong lines inside the range (lambda-dlambda, lambda+dlambda). By default, the minimum emissivity for a listed line is 10^{-18} ph cm^3/s, but this can be modified up or down (minimum value in tables is 10^{-20} ph cm^3/s. Note: identify uses the APEC linelist, which contains lines whose wavelengths may be off by as much as 0.05Å, especially for L shell ions of nickel.

strong : *strong(Temperature, minEmissivity, minLambda, maxLambda])* : Given a temperature (either in Kelvin or keV), and (optionally) a minimum emissivity and wavelength range, the strong routine prints out a list of all strong lines inside the range (minLambda,maxLambda). By default, the wavelength range is from $0 - 10^5$Å, and the minimum emissivity is 10^{-18} ph cm^3/s. The method used to calculate the strong lines is simple to scale the peak emissivity of the line by the ratio of the ion population at the requested temperature to the ion population at the temperature of peak emissivity. This is a relatively crude measure of the line emissivity, and should not be used for any purpose other than finding potential strong lines. The ATOMDB contains explicitly calculated line emissivities for a range of temperatures; these are not used here simply for speed reasons.

describe : *describe(Z, rmJ, lowerlevel, upperlevel)* : Given a transition, labeled by its ion and upper and lower levels, this routine extracts from APED all the data available about the transition and prints them to the screen. The data available depend on what is in APED.

ionbal : *ionbal(Z, rmJ, T)* : Given an ion state Z, rmJ and a temperature T (in Kelvin or in keV), returns the ionization balance at that temperature or temperatures; T can be either a scalar or a verctor.

mdl2latex : *mdl2latex(["filename"])* : mdl2latex outputs a latex table containing data extracted from Sherpa models. The table contains all emission lines fit using a gaussian, lorentzian, or delta function model. The wavelength, flux, and flux error is output, along with a list of the data the line was fit using. Finally, if a line identification has been given by the user (using the modelname lineid "" command), this is output as well. If a filename is given (in quotes), the table is written to the filename instead of the screen.

These commands are primary used to support the previous commands, especially the "describe" command. However, we document them here as they may be useful in writing new *S-lang* routines:

calc_einstein_a : *calc_einstein_a(Z, rmJ, level1, level2, [/silent])* : Given an ion Z, rmJ and two energy levels level1 and level2 (in any order), returns the rate (in Hz) of the transition connecting these levels–if there is one. Otherwise, the routine returns NULL. This routine automatically reads in the necessary wavelength data.

calc_energy : *calc_energy(Z, rmJ, elev, [/silent])* : Given an ion Z, rmJ, and a energy level elev (1 for ground, 2 for the first excited state and so on), this routine returns the energy of the level in eV. The energy level data is automatically read in.

calc_statwt : *calc_statwt(Z, rmJ, elev, [/silent])* : Given an ion Z, rmJ and an energy level elev (1 for ground, 2 for the first excited state and so on), returns the statistical weight for the level. This routine automatically reads in the necessary energy level data.

calc_wavelength : *calc_wavelength(Z, rmJ, level1, level2, [/silent])* : Given an ion Z, rmJ and two energy levels level1 and level2 (in any order), returns the wavelength of the transition connecting these levels–if there is one. Otherwise, the routine returns NULL. This routine automatically reads in the necessary wavelength data.

read_drline : *result = read_drline(Z, rmJ, [/silent])*: Reads the dielectronic recombination data for the appropriate ion Z, rmJ and returns a structure containing the result. If no data can be found, a NULL result is returned.

read_ecoll : *result = read_ecoll(Z, rmJ, [/silent])*: Reads the electron collision data for the appropriate ion Z, rmJ and returns a structure containing the result. If no data can be found, a NULL result is returned.

read_elev : *result = read_elev(Z, rmJ, [/silent])*: Reads the energy level data for the appropriate ion Z, rmJ and returns a structure containing the result. If no data can be found, a NULL result is returned.

read_eline : *result = read_eline(Z, rmJ, [/silent])*: Reads the emission line data for the appropriate ion Z, rmJ and returns a structure containing the result. If no data can be found, a NULL result is returned.

read_ion_bal : *read_ion_bal(filename)* : Reads an ionization balance file from the APED. By default, if no filename is given, the Mazzotta et al. (1998) ionization balance is read. This function has no return value; the ionization balance itself is read into an internal variable. It can be accessed using the ionbal routine described above.

read_pcoll : *result = read_pcoll(Z, rmJ, [/silent])*: Reads the proton collision data for the appropriate ion Z, rmJ and returns a structure containing the result. If no data can be found, a NULL result is returned.