ELSA Documentation for Version 1.0b1

July 6, 2009

Contents

1 Introduction

ELSA is an integrated tool for astrophysics researchers. It is meant to serve a bridge between raw data logs taken with the Image Reduction and Analysis Facility (IRAF) and the final, organized tabular data. It was originally designed to analyze emission-line spectrum data from planetary nebulae, but it is now possible to use **ELSA** on a host of astronomical objects that produce similar spectra. **ELSA** integrates much of the latest research in atomic physics and astrophysics, as well new, original methods, to produce sophisticated calculations of ionic and elemental abundances, electron temperatures, and electron densities based on emission line strengths. It also provides facilities for managing and formatting the resultant data for use in research and publications.

ELSA was developed by Matthew D. Johnson and Jesse S. Levitt at Williams College in the summer of 2005. Large portions of it are based heavily on the work of Karen Kwitter at Williams College and Dick Henry at the University of Oklahoma. It borrows widely from the established work of numerous individuals who have developed atomic data and constants for use in applications like these.

What follows is the comprehensive documentation for **ELSA**. It includes the basic and setup, and operation of the program, as well as a guide for experienced and semi-experienced programmers who wish to modify the code. We would appreciate it if any significant modifications to the code could be brought to our attention, as **ELSA** is still a work in progress and we wish to include as many as improvements and advances as possible in future versions of the code.

If you find **ELSA** to be useful to you in your research and analysis, we encourage you to tell us and provide feedback on how future versions could be expanded and improved.

2 Setup

2.1 Compiling on UNIX Systems

The latest source code and binaries for **ELSA** can always be found on the website:
http://www.williams.edu/astronomy/research/PN/.

To obtain a working copy of **ELSA**, there are two options. If you use Mac OS X (PowerPC), Solaris (SPARC), or Linux (i386), you may download the binaries for your system. These are pre-compiled for specific system and platform. If you're not using one of the above, or you prefer to compile your own binary, obtain the source code distribution. In any case, unpack it as such:

```
gunzip elsa-1.0.tar.gz
tar -xvf elsa-1.0.tar
```

If you're compiling from souce, you then should be able go into the `elsa/src` directory and run `make`. This will build your **ELSA** binary and place it at the root level of the directory (outside `src`. If this seems to have worked correctly, you're done. Now you will probably want to edit your config file, `elsa.conf`, as described in section 3.1.2.

If your system has a non-standard `make` and isn't able to deal with the Makefile provided here, you can build it manually with the following:

```
gcc -o elsa *.c -lm
```

Your C compiler may be called `cc` if your system is old enough not to have the GNU C compiler (GCC) installed on it. If you experience any severe system-specific problems that prevent you from compiling or running **ELSA**, please let us know!

2.2 Other Systems

ELSA is designed on and for UNIX systems. Currently there is no version for Windows or other operating systems. It may be possible to compile and run the code on such systems provided that there is sufficient compatibility with the POSIX library calls. In these cases, you will need to make sure that you link against libmath (the `-lm` option in GCC) or you may experience very strange mathematical behavior.

3 Reference

3.1 General Structure and Configuration

3.1.1 Overview

The source code compiles to a single UNIX binary. This binary will provide an interface to the multiple "tasks" that make up **ELSA**'s suite of tools. Each task has a specific function and takes specific arguments and parameters (typically names of input and output files, or numerical constants required for the operation the task performs). There is also a general configuration file (called `elsa.conf` by default) which provides the values of several important global constants and switches that pertain to all tasks in the program, such as the values of Hα and Hβ, the wavelength tolerance threshold, and more. See section 3.2 for descriptions of these run-time configuration variables.

The tasks fall into one of three categories: log merging and management, table generation, and abundance calculation. Log merging and management tasks process the data in `splot` log files produced by IRAF, and run diagnostics on it to flag unknown wavelengths and measurements with long comments. Table generation tasks will create organized data tables in various table formats after performing corrections for reddening, helium recombination contamination, temperature and density dependence, and Balmer ratio variance. Abundance calculation tasks will compute ionic and elemental abundances, electron temperatures, and electron densities based on line strengths after performing the aforementioned corrections.

ELSA is run from the UNIX command line as such:

```
elsa [options] [task] [task syntax - .]
```

The syntax for each task is listed in its respective subsection below.

There are a small number command line options available. In this version, they only control the preferred output format. This version can output in human readable (tab-delimited) format, in the CSV (comma separated value) format suitable for use with tabular data and plotting programs, and in LaTeX (with the AASTeX extensions) format. Running `elsa -h` will print the online help, which shows the list of tasks and command line options present in the current version. As of the time of this writing, the options are:

`-c` Use a set c (dereddening) value. Only applies to abundance calculation and table generation tasks. Usage: `-c=X.XX`, where `X` is in [0-9].

`-e` Calculate and report error/uncertanties. Applies to table generation and abundance calculation tasks.

`-h` Print online help and exit (no task execution).

`-p` Preserve unparsed lines as comments in `convert_data` task.

`-r` Use human readable output wherever possible.

`-s` Use CSV (comma separated values) output wherever possible.

`-t` Use LaTeX output wherever possible.

`-v` Print version number and exit (no task execution).

It is possible to stack options (such as `-vc`), but as of this version there is no reason to do this as all the options are mutually exclusive. In the event that, for example, two conflicting formatting options are given, the last option given will take precedence. Note that not all output formats are available for all tasks. See the task reference below for more information about possible output formats for specific tasks.

3.1.2 The `elsa.conf` File

The global configuration file is by default named `elsa.conf` and will unpack in the same directory as the source code and documentation. Before running **ELSA**, you should edit the configuration file and make sure that the values in it are acceptable for your data. The configuration file contains comments which explain the values in it. Here are the important parameters you will need to set:

`SpecFile` Contains the name of the file from which the list of "known" or reference wavelengths will be read. See section 3.1.3 for the structure and use of this file. The filename can be an absolute path (beginning with a slash) or a relative path (not beginning with a slash), in which case it is relative to the current working directory when **ELSA** is invoked. Its default is `wavelengths.txt`, which assumes it is a file so named in the same directory as the **ELSA** executable. Must be a string value, but no quotation marks are necessary to delimit it.

`MaxLines` Specifies the maximum number of lines that can be read from a data file (either in `splot` format or ABUN format). **ELSA** will allocate enough memory to hold the number of lines specified by this value, so there is no disadvantage to setting it too high other than that **ELSA** use a larger amount of memory. If it is set lower than the number of lines found in a file, **ELSA** will stop reading lines after it reaches this limit. Must be an integer value.

`MaxSpec` Specifies the maximum number of reference wavelengths that can be read from `SpecFile`. Behavior is similar to that of `MaxLines`, only with regard to the spec file instead of input data files. Must be an integer value.

`MaxTolerance` Determines the upper and lower limits within which **ELSA** will search for wavelengths. This parameter is used in all tasks in **ELSA**. The log management tasks use it to identify "overlapping" wavelengths as described in sections 3.2.1 and 3.2.2. The table generation and abundance calculation tasks use it to determine which reference wavelength a line from the data input file should be associated with. If, for example, this parameter is set to 5.0, observed wavelengths in log files will match the reference wavelength in question if they are within ± 5 angstroms of the reference wavelength. The `splot_flag` task uses this parameter in the same fashion to determine which wavelengths *cannot* be identified with a reference wavelength and should thus be flagged. Must be a floating point value. *(See also the discussion of the radial velocity task parameter in sections 3.2 and 3.3, as it also affects the selection of observed wavelengths to associate with reference wavelengths.)*

`HalphaWavelength` Specifies the reference wavelength for Hα. It should not be listed in the spec file. Must be a floating point value.

`HbetaWavelength` Specifies the reference wavelength for Hβ. It should not be listed in the spec file. Must be a floating point value.

`He2Wavelength` Specifies the reference wavelength for He II. This wavelength can (and probably should) be listed in the spec file. Whereas Hα and Hβ are treated specially and output independently of processing the spec file, He II is not. This wavelength only determines which flux will be used to perform correction for contamination from helium recombination (see section 3.2 for a discussion of this). Once the correction is complete, He II will be processed like the rest of the reference wavelengths.

`HalphaHbetaRatio` Defines the default value for the ratio of Hα/Hβ. Common convention in planetary nebula research to date has fixed this value at 2.86. **ELSA** has a facility to calculate this value based on the temperature and density of the object in question. This parameter's value is used if that facility is turned off, or fails to produce a value for any reason. This value is also used to "seed" the dynamic calculation of the true value. Must be a floating point value.

`BalmerDecrement` This is a boolean switch to tell **ELSA** whether it should use the Balmer Decrement to calculate a correct value for Hα/Hβ. If this is enabled, any log put

through a table generation or abundance calculation routine will require the presence of the O III lines (λ5007 and λ4363) and the S II lines (λ6716 and λ6733) to determine the O III temperature and electron density before calculating the Balmer Decrement. If this presents a problem you can disable this functionality and the aforementioned tasks will only require Hα and Hβ to run. The value of `HalphaHbetaRatio` will be used instead. Must be the string "on" or the string "off" (without quotation marks).

3.1.3 The Spec File

The spec file contains the list of reference wavelengths to be used for table generation and abundance calculation tasks. The format of the file is simple: each line contains a wavelength in angstroms as a floating point value (note that both integers and decimals are acceptable as floating point values) followed by a space, followed by the text description of what ion the wavelength is related to. Everything after the initial space is considered to be this text description. The number of lines in the spec file should not exceed the amount specified by the `MaxSpec` configuration variable. If it does, lines beyond the specified limit are ignored. Comments and blank lines are not allowed in the spec file, and their presence may cause formatting faults which will prevent **ELSA** from being able to complete a run.

The "master list" of wavelengths read from this file is used to generate table output, and to generate the data that is passed to the abundance calculation routines. If a wavelength does not appear in the spec file, it will be ignored by **ELSA** in all table generation and abundance calculation tasks.

See the provided sample "wavelengths.txt" for a guide to creating spec files.

3.2 Log Management Tasks

3.2.1 The `splot_merge` Task

Syntax: `splot_merge <file_1> <file_2> [scale] [output_file]`

The `splot_merge` task consolidates two `splot` log files generated by IRAF into a single log. Raw, unedited logs should work correctly. The expected format of the log files is columns of floating point values. This task will read the files line by line, taking the first column as the wavelength and the third column as the flux. The second column (the continuum level) and all columns after the third are ignored. The task also ignores any line that does not begin with a number (i.e., it will ignore blank lines and "header" lines). Lines beginning with a pound sign (`#`) are taken to be comments. Comments are preserved throughout all processes in **ELSA**. All the text after the pound sign is taken to be the comment. The pound

sign itself and all leading and trailing whitespace are deleted. All comments are associated with the last valid line of data that was read. In the event of multiple consecutive comments, the strings will be concatenated into a single comment string and associated with the last valid line of data.

The output will be in the same format, although zeroes will be output as placeholders for the continuum level (this will not cause problems when the output log is read, since the continuum level is ignored). Comments will be output prefixed by a pound sign following the line that they refer to. Output is written to the named `output_file` if it is provided and the file writable. If not, output will be to the standard output. Since this task has a special output format, all command line options about output formats are ignored.

The parameter `scale` is a floating point value by which all the flux values in `file_2` will be multiplied prior to processing. The goal of the scale factor is normalize all the flux values across both files to the same scale (in the event that the instrumentation used to obtain the data produced different scales for different ends of the spectrum). There is no facility in **ELSA** to calculate the scale factor automatically. The scale factor parameter is optional. If it is not provided, it will be assumed to be 1.0.

After the scale factor has been applied to `file_2`, this task will iterate through each line in both files. It will find all other lines whose wavelength falls within ± the value of the `MaxTolerance` configuration variable of the wavelength in question. All such overlaps that are found will cause you to be prompted to select one of the overlapping lines to be written into the output file. This will eliminate all duplicate lines, whether they happen to be in the same file or in different files. The prompt will present you with the wavelength, flux, and comment for you evaluate.

It is recommended that you check the output file against your input files to ensure that your `MaxTolerance` setting was sufficient to both catch all overlaps, and to prevent finding false overlaps. This task is ultimately intended to produce valid input for the table generation and abundance calculation tasks, which cannot deal with duplicate line matches. Always save original copies of your data.

3.2.2 The `splot_undup` Task

Syntax: `splot_undup <file> [output_file]`

The `splot_undup` task functions very similarly to `splot_merge`. However, instead of reading its "pool" of data from two files, it only reads it from one. There is, obviously, no need for a scale factor in this case. As in `splot_merge`, overlapping measurements are located using the `MaxTolerance` variable and you will be prompted as described in the

previous section to choose one of the overlapping measurements. Output is written to the provided output file name, or to standard output if the output file name is not provided or the file is not writable. As is the case with `splot_merge`, all command line options for output formats are ignored.

3.2.3 The `splot_flag` Task

Syntax: `splot_flag <file> <velocity> [output_file]`

The `splot_flag` task will find measurements in an `splot` log that do not match any of the reference wavelengths in the spec file. It will also find any measurements that have long associated comment strings, which likely indicates some sort of special narrative remark on the measurement (rather than a standard number of colons to indicate uncertainty).

Matching will occur in the inverse of the normal method. The `velocity` parameter will be applied as described in section 3.3.1 and the "expected" wavelengths shifted accordingly. The `MaxTolerance` variable will then be used to determine the upper and lower limits for matching a specific reference wavelength.

Output will be written to the named output file or to standard output if it is not available or not provided. The output will be human readable text in two sections: one for the unknown wavelengths, and one for the long comments.

3.2.4 The `convert_data` Task

Syntax: `convert_data <input_file> [output_file] <format>`

The `convert_data` task will convert wavelength and flux data from an arbitrary table-style format into a format similar to the `splot` log so it can be read by **ELSA**. Data in `input_file` must have a wavelength and a flux on each line, and their position is indicated with the `format` string by a `w` (or `l`) and f, respectively. For example, if each line consists of the wavelength, followed by a comma, followed by the flux, the `format` string should be: `w,f`. The comma here is said to be the delimiter; multiple delimiters in a row (not necessarily the same character, however) will ignore anything in between them. For example: `w,,f` will match `4861, ignored text, 100`, recognizing `4861` as the wavelength and `100` as the flux. Finally, `c` may be used in the `format` string to denote a comment; they are preserved and placed on the line following the data prepended by a `#`, in the `splot` log format. Thus, `c;f,/w` will match `H-beta; 100, garbage / 4861`, reading `H-Beta` as a comment, `4861` as the wavelength, and `100` as the flux, while ignoring `garbage`.

By default, any line that is not successfully parsed is ignored and not put into `output_file`. However, if `convert_data` is run with the `-p` option (for "preserve") all unsuccessfully parsed lines will be preserved as comments. This is not recommended, though, as ELSA considers any comment line in the `splot` log to be a comment for the preceding data.

Finally, `convert_data` can be used for extracting data from TEX tables with the `-t` option. The wavelength is read immediately following the λ (denoted by `$\lambda$`) in the column indicated by the `format` string. (If the sequence `$\lambda$` is not present, it reads wavelength as normal. The other function that the `-t` option performs is to take any colons (:) from the end of the flux and to put them on the following comment line, together with any comments gleaned by use of the `c` token.

For example, if your TEX table looks like this...

Wavelength	**fλ**	**PN 1 Fλ**	**PN 1 Iλ**	**PN 2 Fλ**	**PN 2 Iλ**
He II λ4686	0.04	-	-	1.0:	1.1
Hβ λ4861	0.00	100	100	100	100
[O III] λ5007	-0.04	454	453	1154	1105
Hα λ6563	-0.36	292::	286	420	289

...you would probably want to run the following commands...

```
./elsa -t convert_data myTable.tex PN1.convert.log w&&f
./elsa -t convert_data myTable.tex PN2.convert.log w&&&&f
```

...because the wavelength is before the first ampersand, the flux is after the second one for PN1 and after the fourth for PN2. Note that the 4686 flux would not be recorded in the file `PN1.convert.log`. The `-p` option is strongly discouraged when dealing with TEX files as there would be many formatting lines "preserved" as comments. Finally, the `-t` option has been tested for use with AASTEX's `deluxetable` and will be able to extract data from it.

3.3 Table Generation Tasks

3.3.1 The `splot_table` Task

Syntax: `splot_table <file> <velocity> [output_file]`

The `splot_table` task generates a complete table of extinction-corrected line intensities based on an `splot` log. It takes a single file of input, which must be in the log format described in the previous section. It is strongly recommended that you run `splot_merge` or `splot_undup` on your data before attempting to run this task on it; the output from those tasks will (almost) always be compatible with `splot_table`.

The table generated is based directly on the list of reference wavelengths in the spec file given by the `SpecFile` configuration variable. This task will systematically walk the list of reference wavelengths and try to match them to measured wavelengths in the log file. Two factors apply in this matching. The first is the `velocity` parameter passed to the task from the command line. This is the radial velocity of the object in question, in **kilometers per second**. Based on this parameter, `splot_table` will determine a wavelength shift and apply it to all wavelengths in the file. The equation for this shift is:

$$1 - \frac{v}{c} = \frac{\lambda_{obs}}{\lambda_{ref}} \tag{1}$$

Once this shift has been applied, the task will attempt to match λ_{obs} instead of λ_{ref} to measured wavelengths. Matches will be made for measured wavelengths within the limits given by $\lambda_{obs} \pm$ `MaxTolerance` from the configuration file. If you would prefer not to use the radial shift method, you can simply specify it as 0 on the command line. If you choose to do this, you may need to set `MaxTolerance` to a higher value.

Once all matches have been made, `splot_table` will correct the flux values for interstellar reddening. **ELSA** features some of the most precise techniques for this type of correction currently available. The correction is done by calculating an extinction coefficient c (not to be confused with the speed of light in a vacuum). The equation for this is:

$$c = \frac{1}{0.36} \log \left(\frac{F(H\alpha)}{R_B F(H\beta)} \right) \tag{2}$$

where R_B is the Balmer ratio, or the expected value of Hα/Hβ. If there is a non-zero flux for He II (as specified by the configuration parameter for the He II wavelength), the calculation of c will be done in an iterative fashion to take into account contamination from helium recombination. Additionally, if you have enabled `BalmerDecrement` in the configuration file, this will also be part of the iterative loop. If not, or if there are errors calculating the Balmer decrement, the value of R_B will revert to the value specified by the configuration variable `HalphaHbetaRatio`. Finally, remember that c can be set to a fixed value with the `-c=X.XX` option.

After the calculation of the correction coefficient, the intensity as a fraction of Hβ is calculated using the wavelength-dependent dereddening formula from Savage and Mathis (1979):

$$I = 100 \left(\frac{F}{F(H\beta)} \right) 10^{(1.68 - 4.3\times10^{-4}\lambda + 1.904\times10^{-8}\lambda^2 + 4.839\times10^{-14}\lambda^3)c} \tag{3}$$

Output is written to `output_file` or to the standard output if the file is not available or not

provided. The output file will contain a brief "header" which will display the preliminary c value as well as the observed (uncorrected) fluxes of Hα and Hβ. It will also display the Balmer ratio used in generating the table.

Since this task depends heavily on the presence of Hα and Hβ, it will fail if these lines are not found at the wavelengths specified in the configuration file. If you have enabled dynamic calculation of the Balmer ratio, it will also require the presence of O III lines and S II lines. See section 3.1.2 for more information on this. This task will also fail if multiple matches are found for any given reference wavelength. If this happens and you have already run `splot_merge` or `splot_undup` on your data, you may need to adjust your `MaxTolerance` variable, or examine the data manually and fix the problem.

3.3.2 The `splot_table_composite` Task

Syntax: `splot_table_composite <list_file> [output_file]`

The `splot_table_composite` task performs all the same calculations and corrections as `splot_table`. However, while `splot_table` is intended to produce output containing all available information about a single object, this task will process several logs into a single composite output file. This file will not contain all available information, but rather only the "finished product" (the corrected and scaled intensities relative to Hβ). The goal of this task is to produce output suitable, or nearly suitable, for publication.

The input file, `list_file`, is a text file containing a list of files to be processed. File paths may be absolute or relative. Relative paths are considered relative to the location of the list file, not the directory that **ELSA** is run from. Each file name must be on its own line in the list file, followed by a space, followed by the radial velocity. The radial velocity will be applied as described in the previous section. If you do not wish to use the radial velocity method of wavelength matching, set it to 0 in the list file. It should not be omitted. Assuming there are no errors opening any of the files, their respective data will be read in and processed in the same manner as described in the previous section.

The only output format allowable for this task is LaTeX. Command line output options will be ignored for this task. The LaTeX output will contain a multi-column table (using the AASTeX templates) with the uncorrected and corrected line intensities relative to Hβ. The correction factor, Balmer ratio, and the $\log_{10}$ of the flux of Hβ will be written at the bottom of the table, after the list of reference wavelengths. Comment strings, which are fully preserved throughout **ELSA**, are pared down in this task. Comments are stripped of all characters but colons (as this is the convention for indicating levels of uncertainty used by Kwitter and Henry). The colons will be appended to their associated figures in the final

table to indicate the uncertainty. Comments containing no colons will be discarded entirely.

Note that while there is technically no limitation on how many log files can be put in the list file, it is advisable in our experience not to run more than four or five at a time. More than this may make the table too wide for LaTeX to process correctly into a normal paper size.

3.4 Abundance Calculation

3.4.1 The `splot_abun` Task

Syntax: `splot_abun <file> <velocity> [output_file]`

`splot_abun` is the primary task for calculating abundances. It accepts a single `splot` log, which should ideally be the output from `splot_merge` or `splot_undup`, although it is possible to use raw logs. This task will read in data and perform corrections identically to `splot_table`. Instead of generating a table of the corrected, scaled line intensities, however, `splot_abun` uses them to produce detailed information about abundances, temperatures, and density within the nebula in question.

The details of how these calculations are done are too numerous to explain in this document. The source file `pne_abun.c` contains the function that controls most of this task. Methodology and citations to data can be found in comments in this file. This task is very directly based on ABUN, previously maintained by Richard Henry of the University of Oklahoma.

There are three sections of output from this task. The first section contains abundances of specific ions. These are given as proportions to singly ionized hydrogen. Since a specific ionic temperature is used to calculate each of these abundances, that temperature is listed with each abundance. Comments are preserved here. If multiple lines are used to calculate an abundance, their comments are concatenated and displayed. The second section contains ionic temperatures for selected ions, and the electron density calculated from S II lines. The third section contains general elemental abundances, given as proportions of other elements where appropriate. The corresponding data for the Sun and the Orion Nebula are output for reference purposes.

All three supported output formats are available for this function. However, we recommend CSV or LaTeX output, provided you have the capability to read them (Microsoft Excel can open CSV files). If no format is specified on the command line, this task defaults to CSV output.

3.4.2 The `abun_batch` Task

Syntax: `abun_batch <list_file> [output_file]`

This task invokes the functionality of `splot_abun` over multiple files. The `abun_batch` task does *not* produce a single composite output file. It produces one output file for each input file.

`abun_batch` takes as an input a file containing a list of old-style ABUN input files, one file per line. This is an important difference to note between this task and `splot_abun`. The format of these old-style input files is not the same as the `splot` logs used in most of **ELSA**. The old-style input files consist of a list of wavelengths, followed by their respective **corrected, scaled line intensity**, separated by a space, with one wavelength per line. Once the input is read, the data is considered identical to that generated by the correction and rescaling routines in the tasks that deal with `splot` files. It is also important to note that the wavelengths in these input files must be identical to the reference wavelengths. There is no velocity shifting or matching, only direct comparison between the input wavelengths and reference wavelengths.

The purpose of this function is essentially to maintain backwards compatibility with data that had already been prepared for ABUN. Previously, all corrections and scaling were done by hand or with some other program. The intensities were then placed into these files, directly related to their reference wavelengths, and run through ABUN. However, it is easy enough to generate these input files if you wish to use this task now to work on many objects at once.

As with `splot_abun`, all output options are available for this task, though our recommendation against using "human readable" still stands unless it is a last resort. Output files will be placed in the same directory as the list file. They will be named by appending `.abun.ext` to the basename of the input file (stripping off the portion of the input file name from the first dot onward). The extension depends on the output format. It is `.csv` for CSV files, `.tex` for TEX files, and `.txt` for human readable files.

3.5 ELSA's mini-tasks

3.5.1 The `temp_dens` Task

Syntax: `temp_dens`

The `temp_dens` task allows for a quick, interactive computation of temperatures and densities. Because no external data are provided (i.e. no splot log file is needed), the user

must enter either line intensities or ratios by hand. After running **ELSA**'s `temp_dens` task the user is prompted for a temperatures and densities to calculate. Any number of the supported temperatures and densities listed below may be entered; type the code in the fourth column and press enter (note that there is a space after "`temp`" or "`dens`"). When all desired temperatures and densities have been entered, enter a blank line. Because a temperature is needed to calculate a density, and vice versa, if the set of inputs contains only temperatures, **ELSA** will prompt for a density; likewise, if only densities are entered, a temperature is requested. However, if at least one temperature and at least one density are requested, **ELSA** will run a convergence loop for the first density and temperature input. In any case, the first temperature and density either calculated or entered by the user are used for future calculations. Finally, for each temperature and density requested, **ELSA** will prompt for line strength input. The user may enter either both line strengths, separated by a space, or just the ratio of the two (the prompt specifies which lines are used and which line is the numerator and which is the denominator).

The purpose of this task is to allow for quick temperature and density calculations with the data in hand. This may be useful, for example, with brand new data just received or to check **ELSA**'s temperature and density calculations against previously published results. All temperatures and densities calculated with `temp_dens` are also calculated in **ELSA**'s main abundance calculation routine.

Ion	**Type**	**Lines Used**	**Code**
O II	Temperature	λ7325 / λ3272	`temp O2`
O III	Temperature	λ5007 / λ4363	`temp O3`
N II	Temperature	λ5755 / λ6584	`temp N2`
S II	Temperature	λ4071 / λ6724	`temp S2`
S III	Temperature	λ9532 / λ6312	`temp S3`
Cl II	Density	λ144000 / λ332800	`dens Cl2`
Cl III	Density	λ5517 / λ5537	`dens Cl3`
Ar III	Density	λ89900 / λ218000	`dens Ar3`
S III	Density	λ187000 / λ334800	`dens S3`
Ne V	Density	λ143000 / λ243000	`dens Ne5`

Finally, the intrepid user may want to add temperature and density calculations to this list. This is (hopefully) relatively easy to do. First, the file `elsa_temp_dens.h` in the `src/include` directory must be modified to add the name of the ion, the wavelengths used, and the name of the function to the end of the proper arrays. It should prove relatively straight forward to

follow the examples of the provided temperatures and densities (and remember to update the NUM_TEMP or NUM_DENS defines). Second, the actual temperature or density function must be added; for consistency, it is strongly recommended that it be placed in the file pne_abun.c in the src along with the other temperature and density functions. Also, this allows the direct copying of an existing temperature or density function, which is also highly recommended, as the elsa_get_temp and elsa_get_dens functions are flexible and easy to use. While a basic knowledge of the C language is certainly required, it is hoped that adding new temperatures and densities should not prove too difficult. (See also "Adding New Temperatures and Densities" in the **Programmer's Guide** section.)

3.5.2 The intrat Task

Syntax: intrat

The intrat task provides direct access to **ELSA**'s implementation of Storey and Hummer's (1995) INTRAT routine, which calculates the ratio of emission intensities for two transitions of a hydrogenic ion. At the moment, only H and He II are supported. Like temp_dens, intrat is interactive: it will query the user for element (1 for H, 2 for He), a temperature, a density, an upper level, a lower level, and then upper and lower reference levels. The number returned is the intensity of the "upper level" to "lower level" transition divided by the intensity of the "upper reference level" to "lower reference level" transition.

3.5.3 The normalize Task

Syntax: normalize <optical_file> <UV/IR_file> [temperature] [density]

This task is to be used as an aide when merging two sets of observations, one in the optical, one in the IR or UV, with the splot_merge task. It does not do any data manipulation itself, but rather outputs the merging ratio that should be given to the splot_merge task. There are many requirements for the data fed into this task. First, the temperature and data must either be provided directly or be calculable from the optical_file provided. If no temperature is provided, the O III temperature is used; failing that, the N II temperature; failing that, a default of 10,000. If no density is provided, the S II density is used; failing that, a default of 1,000. Also, the temperature and density arguments do not have to be numbers: they can be strings representing which temperature or density to use. At the moment, the following temperatures are supported: NII, OII, OIII, SII, SIII, as are the following densities: SII, SIII, ClII, ClIII, ArIII, NeV. Note that there is no space

between the element and the ion. If the requested temperature or density are not available, it is treated as if it was not provided. `normalize` functions by using `intrat` to determine the expected ratio of a hydrogenic transition either of He II or H. This means that specific lines are needed for it to work. UV data must have the 1640 line of He II, which will be compared to 4686, and IR data must have either the 7.46μ or 12.4μ line of H, which is compared to Hβ. If the required lines cannot be found, `normalize` fails. If all the required data are present, normalize outputs the ratio.

3.6 The `references` Task

Syntax: `references [output_file] [search_string]`

The `references` task prints out references for atomic data used in **ELSA**. It supports all three forms of output: human-readable displays the output in tab-aligned columns, ideal for quickly locating a reference; CSV outputs the same data into a comma-delimited table, and LaTeX formats the output with AASTeX in two parts: one table for quickly referencing the source of each data point, and again in long citation format as would be included at the end of a paper.

If `output_file` is not provided, output is sent to `stdout`; if `search_string` is not provided, all references are sent to output. References are scanned from the file `references.txt` in the main **ELSA** directory. `references.txt` is produced by `elsa_merge_constants.py`, and includes citations for data from either the "old" data, the TIPbase data, or both, as applicable. (See **Using TIPbase Atomic Constants**, below, for details.)

Using the `search_string` option allows for specific selection of a limited number of references. Query terms should be enclosed in either forward slashes (`//`) for exact phrase matching, curly braces (`{}`) for AND matching, or straight braces (`[]`) for OR matching. References can be matched by anything in **ELSA**'s "reference line;" this includes author, year, publication, ion (e.g. ArIV for Argon^{+3}), and type (collision strength, Einstein A, etc.). For example: `/Collision Strengths/` would find all references containing the phrase "Collision Strength", and thus all collision strength references; `{Mendoza 1983}` would find all references containing both "Mendoza" and "1983"; and `[OII NII]` would find all references containing OII as well as those containing NII. (It would also find all references containing OIII, since OII is contained in OIII. To avoid this, use `[/OII / NII]`, which would match "OII ", and not "OIII".) Finally, search terms can be nested: `{[red /green blue/] yellow}` would match a record containing "yellow" as well as either the phrase "green blue" or the word "red". Theoretically, the nesting limit is infinite; in reality, it is capped by the

recursion limit. We do not anticipate that this limit will be a problem.

3.7 Using TIPbase Atomic Constants

3.7.1 Getting and Using Data from TIPBase

ELSA uses a great deal of atomic data in its abundance calculations and table generating functions. There are, generally speaking, four different types of data needed for each ion: Einstein A values, effective collision strengths, energy levels, and statistical weights. Of these, the Einstein A values and effective collision strengths are particularly laborious to calculate from models, and as such their values are gathered from published literature. The original set of data was compiled by Dick Henry (Kwitter & Henry 2001) and is still available for use. However, we recommend using data from The Iron Project database (see http://vizier.u-strasbg.fr/tipbase/home.html). Since TIPbase is an ongoing process, Python scripts are provided with the source code to download data from TIPbase and merge it with the "old" data. The scripts, `elsa_gettipbaseconstants.py` and `elsa_merge_constants.py`, are located in `elsa/src/include`.

To download TIPbase data, simply go into `elsa/src/include` and type:

```
./elsa_gettipbaseconstants.py
```

To then use TIPbase constants:

```
./elsa_merge_constants.py -t
```

To revert to "old" constants:

```
./elsa_merge_constants.py -o
```

After using `elsa_merge_constants.py` in either form, you must recompile. Move to the `src` directory, and:

```
make clean; make
```

Note that the `make clean` is necessary since `elsa_merge_constants.py` alters header files.

ELSA is distributed using TIPbase constants by default; if you wish to use the old constants you must merge with the `o` option first.

3.7.2 The `table_gen` Task

Syntax: `table_gen`

After getting and merging new atomic data from TIPbase (or switching from TIPbase to old or vice versa)it is recommended that the temperature and density interpolation tables be rebuilt with the `table_gen` task. The `table_gen` task is interactive; after running it the user is asked one by one whether or not to generate each table (default is yes), and for each

table the user is asked for a lower and upper bound in temperature and density (log 10), as well as the step size. Defaults, as well as minimums and maximums, are provided, though not explicitly stated. After generating new tables, **ELSA** should be recompiled. Change to the `src` directory and:

```
make clean; make
```

Again, the `make clean` is necessary because header files have been changed. We realize that this means after updating from TIPbase that two recompiles are necessary; however, updating is needed so infrequently that this should not prove to be too much of a burden.
Hint: Feeling lazy and want to use all the defaults? Try `yes | ./elsa table_gen`.

3.8 List of Recognized Ions

This is a list of all the ions that **ELSA** contains data for and is able to use in abundance calculations, along with which lines **ELSA** scans for each ion and which temperature is used to determine its level populations. This release of **ELSA** only processes optical lines roughly in the range $\lambda 3700 - \lambda 9500$, with a few UV and infrared lines recognized as well. Future plans exist for the inclusion of a large range of infrared and UV lines into the program. If you have specific suggestions or requests for lines or ions to include, don't hesitate to let us know (and point us to a source for the atomic data).

Ion	**Wavelength**	**Temperature**
Ar III	λ7135, λ89900	O III
Ar IV	λ4740	O III
Ar V	λ7005	O III
C III	λ1909	O III
C IV	λ1549	O III
Cl II	λ8578	O III
Cl III	λ5537	O III
Cl IV	λ8045	O III
He II	λ5876	–
He III	λ4868	–
N II	λ6584	N II
N III	λ1751	O III
Ne III	λ3869	O III
Ne IV	λ1602	O III
Ne V	λ1575	O III
O I	λ6300	N II
O II	λ3727, λ7325	N II
O III	λ5007	O III
S II	λ6716, λ6731	N II, S II
S III	λ9069, λ9532	N II, S III
S IV	λ105200	O III

4 Programmer's Guide

Note: This section assumes a reasonable knowledge of the C programming language. It is not within the scope of this document to explain basic programming concepts. Many excellent guides are available on the Internet for this.

4.1 When In Doubt... Copy!

In general, **ELSA** has been designed to be as internally consistent as possible across the different calculations that it makes. That is, the function to calculate the O III temperature looks almost the same as that which calculates the N II temperature, and looks similar to the S III density calculation function, and so on. Thus, the best approach when adding something to **ELSA** is usually to find where it has been done before and copy. If adding a new ionic abundance calculation, look for the other calculations in `pne_abun.c` and copy; if adding a new temperature, copy another `pne_temp` function and change what is needed. In this manner, expansion of **ELSA** is relatively easy.

4.2 General Architecture Notes

Anyone attempting to modify the **ELSA** code should do so with a general idea of the structure of the code. The code is divided up into several source files, each with an associated header file. The basic content of the files is:

`pne_main.c` Contains the `main()` routine and the functions to sort out command line arguments and identify the requested task (if any). No science data is dealt with here.

`pne_read.c` Contains functions to obtain data from files in various ways. This includes the reading of the configuration and spec files, and the standard routine for parsing and reading `splot` files. This file also contains most of the code for calculating extinction correction factors, rescaling line strengths, and other important science tasks.

`pne_task.c` Contains a function to control each task available in the program. If you want to change the behavior of specific functions, this is a good starting point.

`pne_abun.c` Contains the code taken from ABUN and translated from FORTRAN to calculate abundances. Also includes several "helper" functions for this process, and copious comments making reference to the literature that the data used here is taken from.

`pne_output.c` Contains functions to turn commonly used data structures into formatted output.

`pne_func.c` Contains all miscellaneous functions to handle hairier calculations, string formatting, and array management. Some of these are science-related, some are not.

Additionally, the header files `pne_atomic.h`, `pne_ions.h`, `pne_config.h`,and `pne_struct.h` contain common data and declarations that are included by several of the C files.

There several important and commonly used data structures in the program. They are as follows:

`struct LineData` This is a structure that holds all the possible information about a single emission line: wavelength, flux, intensity, energy, and comments. Most task control functions call functions from `pne_read.c` to generate an array of `struct LineData` structures to represent the "data table" contained in a log file. There are three important functions that make these arrays. `pne_get_data` creates an array with as many elements as there were lines in the file (but not more than `MaxLines`) and fills only the `wavelength`, `flux`, and `comment` members of the structures. It returns the count of elements in the array and modifies the array by reference. The function `pne_get_data_with_calcs` returns a pointer to an allocated array of length equal to the length the reference wavelength list. It fills the `wavelength`, `flux`, `intensity`, `energy`, and `comment` members of the array. It proceeds with the calculations using the `HalphaHbetaRatio` variable as the Balmer ratio, but does perform all other corrections for reddening and helium contamination. The function `pne_get_data_with_c_correction` does the exact same thing, but it computes its own Balmer ratio and uses that. Most instances of this structure are referred to by the variable name `m`.

`struct Config` This contains the configuration data read from `elsa.conf` by the function `pne_readconf`. It also contains a substructure of the type `struct LiveData` called `data`. While the normal members of `struct Config` are only modified at the beginning of the program's run, the `data` members are modified by tasks. This is because it is most convenient to keep the radial velocity, and the operative c and Balmer ratio values in a globally accessible structure to eliminate the need to pass them back and forth. The `Config` structure is instantiated at the beginning of the program, and a pointer to this single instance is passed to nearly every function in the program. This avoids truly global variables while still providing a globally-accessible space for certain data. This global instance of this structure is always called `conf`.

`struct ItemData` This holds a single output value, along with information about it. Generally it is used for ionic abundances, and will have the abundance, the uncertainty,

the name of the temperature and density used, and the name of the ion. It also serves as an item in a linked list, having a pointer to another `ItemData` struct. If an ionic abundance is calculated with different temperatures, then they should all be in the same linked list. Likewise, all temperatures and densities are stored in two linked lists (one for temperatures, one for densities). These are read in the output functions and appropriately formatted into tables.

`struct AbunData` This structure holds the output of **ELSA**'s abundance calculations. It has a pointer to an `ItemData` for temperature and density, each of them being the head of a linked list. For abundances, it has the `abundances` two-dimensional array. The first index represents the element, and the second the ion, in spectroscopic notation. Thus, `abundances[8][3]` represents O III abundance, `abundances[6][1]` represents C I abundance. The elemental abundance is stored at the zero position: `abundances[2][0]` holds the He elemental abundance. If an ionic abundance is not calculated or impossible, its position remains null: `abundances[5][5]` will remain `NULL` for the foreseeable future, and `abundances[2][5]`, though it exists in the array, will always be `NULL`. Finally, each element in the `abundances` array is potentially the head of a linked list, and all items in the list will be an abundance for that ion, though calculated differently (usually using a different density or temperature).

These tools should be sufficient for you to add functionality to the program if need be. Use the "get_data" functions to fill your data arrays, and the various functions in `pne_output.c` to get them into useful formats. This is the general flow used in the existing tasks. The technical details of adding new capabilities to **ELSA** follows.

4.3 Adding New Ions

To add a new ionic abundance calculation to **ELSA**, there are a few places that need modification. First, **ELSA** must know to look for the appropriate wavelengths. Edit the spec file (usually `wavelengths.txt` to include the relevant lines. Then open `pne_abun.c` and go to the `pne_abun_calc` function; this is where most of the work will be done. Add any relevant `pne_find_index` calls in the same manner as the existing ones (note that wavelength is taken in **centimeters**. Add declarations for the atomic data needed along with the other atomic data declarations; atomic data is stored in `elsa_ion_constants.h`, and thus declarations should be `extern`. If the atomic data are not present already, they will need to be added: see "Adding New Atomic Data" below. Next the actual calculations for the ion must be added. Ionic calculations are sorted by element further down in the `pne_abun_calc` function. Insert the math in the appropriate place, and create an `ItemData` struct in the

same fashion as the others to hold the output. Add the `ItemData` struct (or rather, a pointer to it) to the appropriate place in the `AbunData` struct's abundances two-dimensional array: the first array index is for the element, the second for the ion (in spectroscopic notation, starting at one, not zero), and if that spot is already occupied, add the new data to the end of the list. The output function will automatically output the new data in the correct place. And remember: when in doubt... copy!

4.4 Adding New Temperatures and Densities

Adding new temperature and density calculations to **ELSA** is also designed to be as easy as possible. First, a new interpolation table must be generated. Open `elsa_table_generators.c` and copy an existing temperature or density function–preferably one that represents the same transitions in a different element or ion–and change the variable and output names to represent the new ion (remember to change the name of the function as well!). Next, open `pne_task.c`, go to the `elsa_table_gen` function, and copy an existing block (containing a `printf`, `fgets`, and `switch` statement), and change the prompt and function call. Finally, go back to `pne_abun.c` and add `#include <include/filename.h>` to the top, where `filename` was entered (or rather, changed) when adding the new interpolation table generation function to `elsa_table_generators.c`. Now, compile and run **ELSA**'s `table_gen` task, remembering that only the newly added table need be generated. If the atomic data is not present, then it will not compile; see "Adding New Atomic Data" below, then try again. Temperature and density functions are in `pne_abun.c`; a good place to start is to copy an existing one and change the variable names used. The interpolation table variables should have been created with the `table_gen` task; the atomic data (Einstein A's, transition energies, etc.) likewise was used in the interpolation table generation, so should also already exist. After the function is in place, go up to `pne_abun_calc` and find the appropriate place to call it; temperature and density are calculated above the abundances. Be sure to insert the new temperature or density in the appropriate linked list in the `AbunData` struct named `Output`; the output function will automatically insert it in the appropriate table. And remember, again: when in doubt... copy!

4.5 Adding New Atomic Data

As new ions, temperatures, and densities are added to **ELSA**'s repertoire, atomic data will be needed to support these calculations. Adding these data is designed to be less intensive on C programming, and more interactive. Ideally, TIPbase has the data needed, and it is only a matter of adding the ions to the list of data to gather from TIPbase to add them

to **ELSA**. Open `elsa_gettipbaseconstants.py`, found in the `include` directory. It is a Python script, but do not fear: first, Python is one of the easiest languages to learn for experienced programmers and the uninitiated alike, and second, no knowledge of Python is actually required. On line 129 is the list of elements, and on 132 is the list of corresponding ions. If a new ion is to be added to an existing element, add it to the appropriate list on the second line (spectroscopic notation); if a new element is to be added, add it to the first list, then add any ions in a new bracketed list in the second line, corresponding to the same position as the new element in the first line. (Note that neither elements nor ions need be in ascending order.) Save the script, and run it, noting whether or not it was able to get the data for the new ion. There are two categories: energy/Einstein A and collision strength. If either of them fail, data need to be manually entered into `elsa_old_constants.h`. Because a five-level atom model is used, the arrays are of length 10. The first element represents the datum for the two to one transition, the second for the three to one, the third for the four to one, and so on: the fifth is the three to two transition, the sixth the four to two, and the tenth the five to four. If this ordering seems counter-intuitive, it is a relic of the Fortran to C conversion. In any case, the next script to be run is `elsa_merge_constants.py`. It will recognize that there are no statistical weight data for the new ion, and query the user for it. It will also point out if there is lacking collision strength data, and direct the user to enter it. After the merge script is run, the data should be available in the same manner as the other atomic data, ready for use. Finally, we recommend adding any references used for Einstein A's or collision strengths to the file `old_references.txt` in the include directory (using the same format as the existing ones) and running the merge script again. This will update `references.txt` in the main directory and add them to any references output by the `references` task.

4.6 Adding Configuration Variables

Begin by editing `pne_struct.h` and adding a new member to the `Config` structure to represent the new variable you wish to add. Next, edit `pne_read.c` and add a new `if` statement to the series of them in the `pne_readconf` function. Use the string comparison library call to identify your parameter keyword in the config file (comparing with the local variable `param`). You may then use the `value` local variable as you fit. If you are trying to read a numerical value, make sure to call `strtod` or `strtol` on it. If it is a string, it will suffice to use `strcpy`.

5 Acknowledgments

We are grateful for the help, support, and enthusiasm that Professors Kwitter and Henry have contributed to this project. We also thank the multitude of individuals who have published the data and constants necessary for the processes in this program to work. Finally, we knowledge the continued financial support of the Keck Northeast Astronomy Consortium and the National Science Foundation in making the research and development of **ELSA** possible. This research is partially supported by NSF grant AST 03-07118 to the University of Oklahoma.

6 Authors' Contact Information

Matthew D. Johnson
Department of Astronomy, Wesleyan University
Middletown, Connecticut 06459
mdjohnson@@wesleyan.edu

Jesse S. Levitt
Department of Astronomy, Williams College
Williamstown, Massachusetts 01267
Jesse.S.Levitt@@williams.edu

Peter J. J. O'Malley
Department of Physics and Astronomy, Haverford College
Haverford, Pennsylvania 19041
pomalley@@haverford.edu