University of Central Florida

Transit Code Manual

A Radiative-Transfer Code for Planetary Atmospheres

Authors:
Ryan Challener
Patricio Cubillos
Jasmina Blecic

Supervisor:
Dr. Joseph Harrington

9 July 2015

1 Introduction IN PROGRESS

This document is intended to provide the `transit` user with the information necessary to modify the code for their own uses. It describes all headers, custom variable types, structures, and functions used within the `transit` program. We also include information about which functions allocate, fill in, and free data arrays.

The `transit` program is structure-oriented. All the variables used by `transit` are held in structures, as listed in Section 5. These structures are passed around to functions which allocate, fill out, and free the variables within the structures. In Section 5 all arrays within the program are cross-referenced with the functions that alter them.

The functions of `transit` are split into two chunks: initialization and calculation. The initialization functions set up the variables and structures that are needed to make the radiative transfer calculations. This includes parsing the configuration file, making sampling arrays, reading the input files (line transition information, collision-induced absorption), and creating the opacity grid. The calculation functions make the extinction and optical depth calculations, for both transit and eclipse geometry, and create the output files. Figure 1 shows the function structure of `transit`.

In Section 2 we describe all the headers included in `transit`. Section 3 gives all the custom variable types and their purpose. Then in Section 4 we list all the defined constants in `transit`. Section 5 lists both the structures and structure types used, and makes note of which function alters each array. In Section 6 we list all source files and functions, and provide a list of the variables which are altered by each function along with a walkthrough of each major function. Finally, Section **??** lists some equations that are important to the calculations made in `transit`.

2 List of Headers used by `transit` IN PROGRESS

This section lists all headers included in the `transit` program. Those which are unique to `transit` can be found in the transit/include/ directory. The file transit.h contains the lines which include the rest of the headers. transit.h:

Contains common headers for the `transit` program and defines some functions.

stdarg.h:

Allows functions to accept an indefinite number of arguments.

math.h:

Contains common mathematical operations.

errno.h:

Defines the `errno` macro, allowing errors to be checked.

sys/types.h:

Defines a large number of types.

sys/stat.h:

Defines the structure of data returns by stat() functions.

sys/time.h:

Defines a number of time-related structures, functions, and arguments.

unistd.h:

Defines many miscellaneous contanst, types, and functions.

sampling.h:

Declares resampling and interpolation functions.

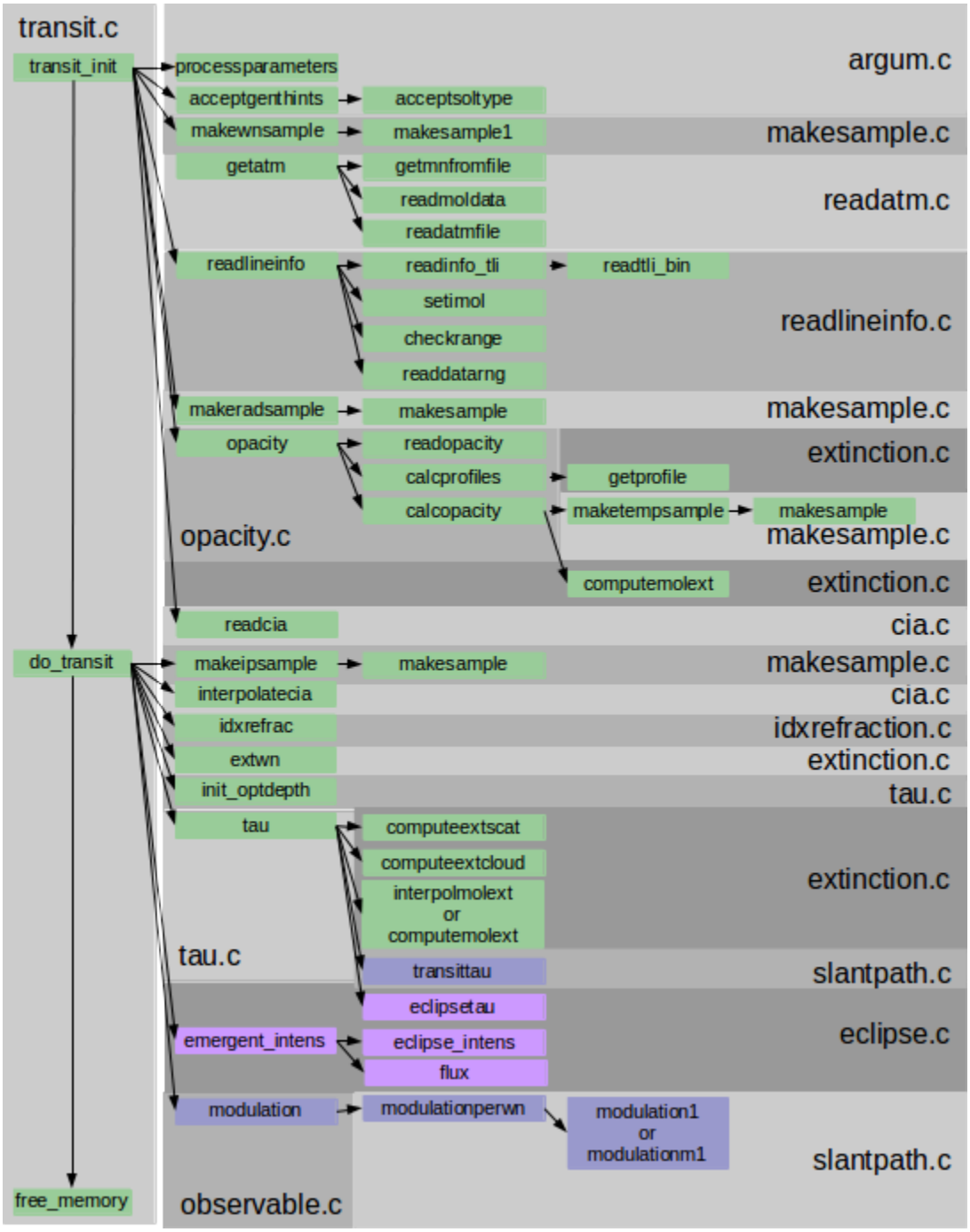

Figure 1: `transit` function structure. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. For example, at the beginning, transit_init calls processparameters, then transit_init calls acceptgenhints, then acceptgenhints calls acceptsoltype, and so on. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geomtry, and green functions are used in both. The shaded backgrounds indicate where each function can be found.

profile.h:
Declares Voigt profile calculation functions.
iomisc.h:
Declares miscellaneous input/output function (in iomisc.c).
numerical.h:
Declares functions found in numerical.c.
xmalloc.h:
Redefines malloc(), realloc(), and calloc().
strings.h:
Defines several string manipulation functions.
stdlib.h:
Defines many general purpose functions.
stdio.h:
Defines many general input/output functions.
alloca.h:
Defines function alloca(), which allocates memory which is freed when the function that called alloca() returns to its caller.
flags_tr.h:
Defines `transit` flags.
constants_tr:
Defines `transit` constants (see Table 2).
types_tr:
Defines `transit` custom variable types (see Table 1).

3 List of Custom Variable Types in `transit` DONE

Table 1 lists the user-defined data types used in Transit. These declarations are located in transit/transit/include/types_tr.h.

Table 1: Custom Variable Types

Name	Type	Description
PREC_NSAMP	int	Radius and wavelength indices
PREC_NREC	long	Record indices
PREC_ZREC	double	Partition-function data
PREC_LNDATA	double	Line-transition data
PREC_RES	double	Partial result
PREC_ATM	double	Atmospheric data
PREC_CS	double	Cross-section data
PREC_CIA	double	Collision-induced absorption data
PREC_VOIGT	float	Voigt profile data type
PREC_VOIGTP	double	Voigt profile data type

(FINDME: When is PREC_VOIGTP used vs PREC_VOIGT?)

4 List of Constants in `transit` DONE

Table 2 lists the user-defined constants used in `transit`. The definitions can be found in transit/transit/include/constants_tr.h.

Table 2: Constants

Name	Value	Description
RHOSTP	1.29e-3 g cm^{-3}	Density at STP
PI	3.141592653589793	Pi
DEGREES	PI/180.0	Radians per degree
GGRAV	6.673e-8 erg cm g^2	Gravitational constant
HOUR	3600.0 s	Seconds per hour
AU	14959786896040.492 cm	Centimeters per AU
ANGSTROM	1e-8 cm	Centimeters per angstrom
SUNMASS	1.9891e33 g	Mass of the sun
SUNRADIUS	6.95508e10 cm	Radius of the sun
AMU	1.66053886e-24 g	Grams per atomic mass unit
LO	2.686763e19 cm^{-3}	Loschmidt constant
EC	4.8032068e-10 statC	Electron charge
LS	2.99792458e10 cm s^{-1}	Light speed
ME	9.1093897e-28 g	Mass of an electron
KB	1.380658e-16 erg K^{-1}	Boltzmann constant
H	6.6260755e-27 erg s	Planck constant
HC	H*LS erg cm	Planck constant × speed of light
SIGCTE	(PI*EC^2)/(LS^2*ME*AMU) cm g^{-1}	Cross-section constant
EXPCTE	(H*LS)/KB cm K	Exponent constant
ONEOSQRT2PI	0.3989422804	1/sqrt(2pi)
SQRTLN2	0.83255461115769775635	sqrt(ln(2))
MAXNAMELEN	20	Maximum length of name strings

5 List of Structures in the `transit` Files IN PROGRESS

5.1 Structure Types

Sampling properties of impact parameter, wavenumber, etc.

```
typedef struct {
  PREC_NREC n;      /* Number of elements                          */
  PREC_RES d;       /* Spacing                                     */
  PREC_RES i;       /* Initial value                               */
  PREC_RES f;       /* Final value                                 */
  int o;            /* Oversampling                                */
  PREC_RES *v;      /* Values of the sampling                      */
    /* ALLOCATED:       getatm                                     */
    /* ALLOCATED:       readatmfile                                */
    /* FILLED OUT:      readatmfile                                */
```

```
    /* FILLED OUT:      radpress                                    */
    /* FREED:           freemem_samp                                */
  double fct;       /* v units factor to cgs                       */
} prop_samp;
```

Isotopes' variable information.

```
typedef struct {
  unsigned int n;   /* Arrays' length                              */
  double *z;        /* Partition function [radius or temp]         */
    /* ALLOCATED:       readtli_bin                                 */
    /* FILLED OUT:      readtli_bin                                 */
    /* FREED:                                                       */
} prop_isov;
```

Isotopes' fixed information.

```
typedef struct {
  int d;            /*  Database to which they belong */
  char *n;          /*  Isotope name                  */
  PREC_ZREC m;      /*  Isotope mass                  */
} prop_isof;
```

Molecule properties.

```
typedef struct{
  int n;           /*  Number of elements      */
  PREC_ATM *d;     /*  Density   [n]           */
    /* ALLOCATED:       getatm                                      */
    /* ALLOCATED:       readatmfile                                 */
    /* FILLED OUT:                                                  */
    /* FREED:                                                       */
  PREC_ATM *q;     /*  Abundance [n]           */
    /* ALLOCATED:       getatm                                      */
    /* ALLOCATED:       readatmfile                                 */
    /* FILLED OUT:                                                  */
    /* FREED:                                                       */
} prop_mol;
```

Atmosphere properties.

```
typedef struct {
  double *mm;       /*  Mean molecular mass [rad]        */
    /* ALLOCATED:       getatm                                      */
    /* ALLOCATED:       readatmfile                                 */
    /* FILLED OUT:                                                  */
    /* FREED:                                                       */
```

```
  PREC_ATM *p;      /*  Pressure    [rad]               */
    /* ALLOCATED:       getatm                                    */
    /* ALLOCATED:       readatmfile                               */
    /* FILLED OUT:                                                */
    /* FREED:                                                     */
  PREC_ATM *t;      /*  Temperature [rad]               */
    /* ALLOCATED:       getatm                                    */
    /* ALLOCATED:       readatmfile                               */
    /* FILLED OUT:                                                */
    /* FREED:                                                     */
  PREC_ATM pfct;    /*  p units factor to cgs (dyne/cm2) */
  PREC_ATM tfct;    /*  t units factor to cgs (Kelvin)   */
} prop_atm;
```

Database properties.

```
typedef struct {
  char *n;          /*  Database name                             */
  char *molname;    /*  Molecule name                             */
  unsigned int i;   /*  Number of isotopes                        */
  int s;            /*  Cumulative first isotope's index          */
} prop_db;
```

Database temperatures.

```
typedef struct {
  unsigned int t;   /* Number of temperatures                     */
  double *T;        /* Temperatures                               */
    /* ALLOCATED:       readtli_bin                               */
    /* FILLED OUT:                                                */
    /* FREED:                                                     */
} prop_dbnoext;
```

Ray solution parameters and integrator functions.

```
typedef struct {
  const char *name;          /* Ray solution name                 */
  const char *file;          /* Ray solution source file          */
  const short monospace;     /* Request equispaced inpact parameter? */
  PREC_RES (*optdepth)       /* Extinction-coefficient integrator function */
       (struct transit *tr,
        PREC_RES b,              /* Height of ray path            */
        PREC_RES *ex);           /* Extinction array [rad]        */
  PREC_RES (*spectrum)       /* Optical-depth integrator function */
        (struct transit *tr,
         PREC_RES *tau,          /* Optical depth                 */
         PREC_RES w,             /* Wavenumber value              */
```

```
        long last,              /* index where tau exceeded toomuch        */
        PREC_RES toomuch,       /* Cutoff optical depth                    */
        prop_samp *r);          /* Impact parameter or layers' radius      */
} ray_solution; transit.h:
```

5.2 Structures

Proportional-abundance isotopic parameters.

```
struct atm_isoprop{
  double f;            /* Fractional abundance                          */
  double m;            /* Isotope mass                                  */
  int eq;              /* Isotope index from transit.ds.isotopes        */
  char n[maxeisoname]; /* Isotope name                                  */
  char t[maxeisoname]; /* Molecule name                                 */
};
```

Line transition parameters

```
struct line_transition{
  PREC_LNDATA *wl;       /* Wavelength                                  */
    /* ALLOCATED:       readdatarng                                     */
    /* FILLED OUT:      readdatarng                                     */
    /* FREED:           freemem_linetranstion                           */
  PREC_LNDATA *elow;     /* Lower-state energy                          */
    /* ALLOCATED:       readdatarng                                     */
    /* FILLED OUT:      readdatarng                                     */
    /* FREED:           freemem_linetranstion                           */
  PREC_LNDATA *gf;       /* gf value                                    */
    /* ALLOCATED:       readdatarng                                     */
    /* FILLED OUT:      readdatarng                                     */
    /* FREED:           freemem_linetransition                          */
  short *isoid;          /* Isotope ID (Assumed to be in range)         */
    /* ALLOCATED:       readdatarng                                     */
    /* FILLED OUT:      readdatarng                                     */
    /* FREED:           freemem_linetransition                          */
  double wfct;           /* wl units factor to cgs                      */
  double efct;           /* elow units factor to cgs                    */
};
```

Line information parameters

```
struct lineinfo{
  struct line_transition lt; /* Line transitions                        */
  unsigned short tli_ver;    /* TLI version                             */
  unsigned short lr_ver;     /* lineread version                        */
  unsigned short lr_rev;     /* lineread revision                       */
```

```
  prop_samp wavs;            /* Wavelength sampling                          */
  double wi, wf;             /* Initial and final wavelength in database     */
  long endinfo;              /* Position at the end of the info part
                                of the info file                             */
  int asciiline;             /*  TLI line of first transition.
                                Zero if a binary file.                       */
  int ni;                    /* Number of isotopes                           */
  int ndb;                   /* Number of databases                          */
  prop_isov *isov;           /* Variable isotope information (w/temp) [iso] */
    /* ALLOCATED:       readtli_bin                                          */
    /* FILLED OUT:                                                           */
    /* FREED:           freemem_lineinfo                                     */
  prop_dbnoext *db;          /* Temperature info from databases [DB]         */
    /* ALLOCATED:       readtli_bin                                          */
    /* FILLED OUT:                                                           */
    /* FREED:           freemem_lineinfo                                     */
  PREC_NREC n_l;             /* Number of lines in database                  */
};
```

Atmospheric data file parameters.

```
struct atm_data{
  int n_aiso;             /* Number of molecules in atmosphere file          */
  prop_samp rads;         /* Radius sampling                                 */
  prop_atm atm;           /* Atmospheric properties                          */
  prop_mol *molec;        /* Molecular information [n_aiso]                  */
    /* ALLOCATED:       getatm                                               */
    /* FILLED OUT:                                                           */
    /* FREED:                                                                */
  double *mm;             /* Mean molecular mass [rad]                       */
    /* ALLOCATED:       getatm                                               */
    /* FILLED OUT:                                                           */
    /* FREED:                                                                */
  char *info;             /* Optional atmosphere file information or label   */
  _Bool mass;             /* Abundances in isov by mass (1) of by number (0) */
  int begline;            /* Line of first radius dependent info             */
  long begpos;            /* Position of first radius dependent info         */
};
```

Extinction array and extinction parameters.

```
struct extinction{
  PREC_RES **e;      /* Extinction value [rad][wav]                          */
    /* ALLOCATED:                                                            */
    /* FILLED OUT: extinction                                                */
    /* FREED:                                                                */
  int vf;            /* Number of fine—bins of the Voigt function           */
```

```
  float ta;          /* Number of alphas that have to be contained in
                        the profile                                    */
  _Bool *computed;   /* Whether the extinction at the given radius was
                        computed [rad]                                 */
  double ethresh;    /* Lower extinction-coefficient threshold         */
};
```

Opacity array and opacity parameters.

```
struct opacity{
  PREC_RES ****o;         /* Opacity grid [temp][iso][rad][wav]        */
    /* ALLOCATED:                                                     */
    /* FILLED OUT: extinction
*/
    /* FREED:                                                         */
  PREC_VOIGT ***profile;  /* Voigt profiles [nDop][nLor][2*profsize+1] */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED:                                                         */
  PREC_NREC **profsize;   /* Half-size of Voigt profiles [nDop][nLor]  */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED:                                                         */
  double *aDop,           /* Sample of Doppler widths [nDop]           */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED:                                                         */
       *aLor;             /* Sample of Lorentz widths [nLor]           */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED:                                                         */
  PREC_RES *temp,         /* Opacity-grid temperature array            */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED:                                                         */
       *press,            /* Opacity-grid pressure array               */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED:                                                         */
       *wns;              /* Opacity-grid wavenumber array             */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED:                                                         */
  PREC_ATM **ziso;        /* Partition function per isotope [niso][Ntemp] */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED:                                                         */
```

```
  int *molID;             /* Opacity—grid molecule ID array                 */
    /* ALLOCATED:                                                          */
    /* FILLED OUT:                                                         */
    /* FREED:                                                              */
  long Nwave, Ntemp, Nlayer, Nmol, /* Number of elements in opacity grid    */
      nDop, nLor;         /* Number of Doppler and Lorentz—width samples    */
};
```

Index of refraction array.

```
struct idxref{
  PREC_RES *n;   /* Index of refraction [rad]                              */
    /* ALLOCATED:                                                          */
    /* FILLED OUT:                                                         */
    /* FREED:                                                              */
};
```

Save file information.

```
#if 0
struct savefiles {
  char *ext;         /* saves extinction            */
  char *tau;         /* after tau() savefile        */
  char *modulation;  /* after modulation() savefile */
};
#endif
```

Optical depth array and related information.

```
struct optdepth{
  PREC_RES **t;     /* Optical depth [wn][ip]                              */
    /* ALLOCATED:       init_optdepth                                      */
    /* FILLED OUT:      tau                                                */
    /* FREED:           freemem_tau                                        */
  long *last;       /*  Level index where the optical depth reached toomuch
                       (counting from the top of the atmosphere) [wn]      */
    /* ALLOCATED: init_optdepth                                            */
    /* FILLED OUT: tau                                                     */
    /* FREED: freemem_tau                                                  */
  double toomuch;   /* Optical depth values greater than this won't be
                       calculated: the extinction is assumed to be zero.   */
};
```

Intensity array.

```
struct grid{
  PREC_RES **a;      /* Intensity grid, 2D, [an][wnn]                      */
    /* ALLOCATED:                                                          */
```

```
    /* FILLED OUT:                                                    */
    /* FREED:                                                         */
};
```

Information about the geometry of the transit or eclipse.

```
struct geometry{
  float smaxis;       /* Semimajor axis                               */
  double smaxisfct;   /* 'smaxis' times this gives cgs units.         */
  double time;        /* this value is 0 when in the middle of the eclipse  */
  double timefct;     /* 'time' times this gives cgs units            */
  float incl;         /* inclination of the planetary orbit with respect
                         to the observer, 90 degrees is edge on       */
  float inclfct;      /* Units to convert inclination to radians      */
  double ecc;         /* eccentricty                                  */
  double eccfct;      /* eccentricity's units                         */
  double lnode;       /* longitude of the ascending node              */
  double lnodefct;    /* longitude of the ascending node units        */
  double aper;        /* argument of the pericenter                   */
  double aperfct;     /* argument of the pericenter units             */

  double starmass;    /* Mass of the star                             */
  double starmassfct; /* 'starmass' times this gives cgs units.       */

  double starrad;     /* Star's radius                                */
  double starradfct;  /* 'starrad' times this gives cgs units.        */

  double x, y;        /*  Coordinates of the center of the planet with
                         respect to the star. 'fct' to convert to cgs is
                         found in rads.fct. These fields are not hinted.  */

  _Bool transpplanet; /* If true, set maximum optical depth to toomuch  */
};
```

Isotope information.

```
struct isotopes{
  prop_isof *isof;    /* Fixed isotope information      [n_i]         */
    /* ALLOCATED:       readtli_bin                                   */
    /* FILLED OUT:      readtli_bin                                   */
    /* FREED:           freemem_isotopes                              */
  prop_isov *isov;    /* Variable isotope information   [n_i]         */
    /* ALLOCATED:       readtli_bin                                   */
    /* FILLED OUT:      readdatarng                                   */
    /* FREED:           freemem_isotopes                              */
  double *isoratio;   /* Isotopic abundance ratio       [n_i]         */
    /* ALLOCATED:       readtli_bin                                   */
    /* FILLED OUT:      readtli_bin                                   */
    /* FREED:           freemem_isotopes                              */
```

```
  int *imol;          /* Molecule index for this isotope[n_i]              */
    /* ALLOCATED:                                                          */
    /* FILLED OUT:                                                         */
    /* FREED:           freemem_isotopes                                   */
  prop_db *db;        /* Database's info [n_db]                            */
    /* ALLOCATED:       readtli_bin                                        */
    /* FILLED OUT:      readtli_bin                                        */
    /* FREED:           freemem_isotopes                                   */
  int n_db,           /* Number of databases                               */
      n_i,            /* Number of isotopes                                */
      nmol;           /* Number of different molecules having a line list  */
};
```

Molecule information.

```
struct molecules{
  int nmol;           /* Number of molecules                               */
  prop_mol *molec;    /* Molecular properties                              */
    /* ALLOCATED:       getatm                                             */
    /* FILLED OUT:                                                         */
    /* FREED:                                                              */
  char **name;        /* Molecules' names                                  */
    /* ALLOCATED:       getmnfromfile                                      */
    /* FILLED OUT:      getmnfromfile                                      */
    /* FREED:                                                              */
  PREC_ZREC *mass;    /* Molecules' masses                                 */
    /* ALLOCATED:                                                          */
    /* FILLED OUT:      getmoldata                                         */
    /* FREED:                                                              */
  PREC_ZREC *radius;  /* Molecules' radii                                  */
    /* ALLOCATED:       getatm                                             */
    /* FILLED OUT:      getmoldata                                         */
    /* FREED:                                                              */
  int *ID;            /* Molecule universal ID                             */
    /* ALLOCATED:       getatm                                             */
    /* FILLED OUT:      getmoldata                                         */
    /* FREED:                                                              */
};
```

Flux

```
struct outputray{
  PREC_RES *o;     /* Output as seen before interaction with telescope     */
    /* ALLOCATED:                                                          */
    /* FILLED OUT:                                                         */
    /* FREED:                                                              */
};
```

Cloud extinction information.

```
struct extcloud{
  double cloudext;     /* Maximum opacity in [cm-1]                          */
  double cloudtop;     /* Radius at which clouds start                       */
  double cloudbot;     /* Radius at which clouds has it maximum thickness
                          'cloudext'.                                        */
};
```

Scattering extinction information.

```
struct extscat{
  double prm;
};
```

Saved extinction grid name.

```
struct saves{
  char *ext;   /* Extinction grid                                           */
};
```

Stores requested extinction, optical depth, or CIA detailed information.

```
struct detailfld{
  int n;         /* Number of requested wavenumber samples */
  PREC_RES *ref; /* Array of wavenumbers requested         */
    /* ALLOCATED:                                                           */
    /* FILLED OUT:                                                          */
    /* FREED:                                                               */
  char file[80]; /* Output filename                        */
  char name[30]; /* Name of field                          */
};
```

Detailed output for extinction, optical depth, or CIA.

```
struct detailout{
  struct detailfld ext, tau, cia;
};
```

Collision-Induced Absorption (CIA) extinction information.

```
struct cia{
  int nfiles;       /* Number of CIA files                                  */
  PREC_CIA **e;     /* Extinction from all CIA sources [wn][temp]           */
    /* ALLOCATED:                                                           */
    /* FILLED OUT: interpolatecia                                           */
    /* FREED: freemem_cia                                                   */
  PREC_CIA ***cia;  /* Tabulated CIA extinction      [nfiles][nwave][ntemp] */
    /* ALLOCATED:                                                           */
    /* FILLED OUT:                                                          */
    /* FREED: freemem_cia                                                   */
  PREC_CIA **wn;    /* Tabulated wavenumber  arrays  [nfiles][nwave]        */
    /* ALLOCATED:                                                           */
```

```
    /* FILLED OUT:                                                    */
    /* FREED: freemem_cia                                             */
  PREC_CIA **temp;  /* Tabulated temperature arrays  [nfiles][ntemp]  */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED: freemem_cia                                             */
  int *nwave;       /* Number of wavenumber samples  [nfiles]         */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED: freemem_cia                                             */
  int *ntemp;       /* Number of temperature samples [nfiles]         */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED: freemem_cia                                             */
  int *mol1, *mol2; /* Pairs of molecule's ID        [nfiles]         */
    /* ALLOCATED:                                                     */
    /* FILLED OUT:                                                    */
    /* FREED: freemem_cia                                             */
};
```

Structure containing all user-given information that is passed to the transit structure upon approval.

```
struct transithint{
  char *f_atm,          /* Atmosphere filename                       */
       *f_line,         /* TLI filename                              */
       *f_opa,          /* Opacity filename                          */
       *f_out,          /* Output (main) filename                    */
       *f_toomuch,      /* Output toomuch filename                   */
       *f_outsample,    /* Output sample filename                    */
       *f_molfile;      /* Known molecular info filename             */
  PREC_NREC ot;         /* Radius index at which to print output from tau   */
  prop_samp rads, ips,  /* Sampling properties of radius, impact parameter, */
       wavs, wns, temp;   /* wavelength, wavenumber, and temperature  */
  char *angles;         /* String with incident angles (for eclipse)  */
  char *qmol, *qscale;  /* String with species scale factors          */
  float allowrq;        /* How much less than one is accepted, and no warning
                           is issued if abundances don't ad up to that      */
  float timesalpha;     /* Number of alphas that have to be contained in a
                           calculated profile, one side only          */
  int voigtfine;        /* Fine-binning for Voigt function in kapwl(), if
                           accepted it goes to tr.ds.op.vf             */
  int nDop, nLor;       /* Number of broadening width samples          */
  float dmin, dmax, lmin, lmax; /* Broadening-width samples boundaries */
  int verbnoise;        /* Noisiest verbose level in a non debugging run    */
  _Bool mass;           /* Whether the abundances read by getatm are by
                           mass or number                             */
  _Bool opabreak;       /* Break after opacity calculation flag        */
```

```
  long fl;              /* flags                                          */
  _Bool userefraction;  /* Whether to use variable refraction             */
  double p0, r0;        /* Pressure and radius reference level            */
  double gsurf;         /* Surface gravity                                */

  double toomuch;       /* Optical depth values greater than this won't be
                           calculated: the extinction is assumed to be zero  */
  int taulevel;         /* Tau integration level of precision             */
  int modlevel;         /* Modulation integration level of precision      */
  char *solname;        /* Name of the type of solution                   */
  struct geometry sg;   /* System geometry                                */
  struct saves save;    /* Saves indicator of program stats               */

  struct extcloud cl;
  struct detailout det;

  double ethresh;       /* Lower extinction—coefficient threshold         */
  char **ciafile;
  int ncia;
};
```

Main data structure.

```
struct transit{
  char *f_atm,       /* Atmosphere filename                               */
       *f_line,      /* TLI filename                                      */
       *f_opa,       /* Opacity filename                                  */
       *f_out,       /* Output (main) filename                            */
       *f_toomuch,   /* Output toomuch filename                           */
       *f_outsample, /* Output sample filename                            */
       *f_molfile;   /* Known molecular info filename                     */
  PREC_NREC ot;      /* Radius index at which to print output from tau    */

  FILE *fp_atm, *fp_opa, *fp_out, *fp_line; /* Pointers to files          */
  float allowrq;    /* How much less than one is accepted, so that no warning
                       is issued if abundances don't ad up to that       */
  PREC_RES telres;  /* Telescope resolution                              */
  long int angleIndex; /* Index of the current angle                     */
  prop_samp rads,      /* Sampling properties of radius,                 */
    /* ALLOCATED:                                                        */
    /* FILLED OUT:      makeradsample                                    */
    /* FREED:           freemem_samp                                     */
           ips,       /* impact parameter,                              */
    /* ALLOCATED:                                                        */
    /* FILLED OUT:      makeipsample                                     */
    /* FREED:           freemem_samp                                     */
           owns,      /* oversampled wavenumber,                        */
    /* ALLOCATED:                                                        */
```

```
    /* FILLED OUT:      makewnsample                                    */
    /* FREED:           freemem_samp                                    */
            wavs,      /* wavelength,                                   */
    /* ALLOCATED:                                                       */
    /* FILLED OUT:                                                      */
    /* FREED:           freemem_samp                                    *
            wns,       /* wavenumber,                                   */
    /* ALLOCATED:                                                       */
    /* FILLED OUT:      makewnsample                                    */
    /* FREED:           freemem_samp                                    */
            temps;     /* temperature                                   */
    /* ALLOCATED:                                                       */
    /* FILLED OUT:      maketempsample                                  */
    /* FREED:           freemem_samp                                    */
  prop_atm atm;      /* Sampled atmospheric data                        */
  _Bool opabreak;    /* Break after opacity calculation                 */
  int ndivs,         /* Number of exact divisors of the oversampling factor */
     *odivs;         /* Exact divisors of the oversampling factor       */
    /* ALLOCATED:                                                       */
    /* FILLED OUT:      makewnsample
*/
    /* FREED:                                                          */
  int voigtfine;     /* Number of fine-bins of the Voigt function       */
  float timesalpha;  /* Broadening profile width in number of Doppler or
                        Lorentz half width                             */
  double p0, r0;     /* Pressure and radius reference level             */
  double gsurf;      /* Surface gravity                                 */
  int ann;           /* Number of angles                                */
  double *angles;    /* Array of incident angles for eclipse geometry   */
    /* ALLOCATED:                                                      */
    /* FILLED OUT:                                                     */
    /* FREED:                                                          */
  int nqmol;         /* Number of species scale factors                 */
  double *qscale;    /* Species scale factors                           */
    /* ALLOCATED:                                                      */
    /* FILLED OUT:                                                     */
    /* FREED:                                                          */
  int *qmol;         /* Species with scale factors                      */
    /* ALLOCATED:                                                      */
    /* FILLED OUT:                                                     */
    /* FREED:                                                          */
  int taulevel;      /* Tau integration level of precision              */
  int modlevel;      /* Modulation integration level of precision       */

  long fl;           /* flags                                           */
  long interpflag;   /* Interpolation flag                              */
  long pi;           /* progress indicator                              */
```

```
  ray_solution *sol; /* Transit solution type                                */
  PREC_RES *outpret; /* Output dependent on wavelength only as it travels
                        to Earth before telescope                           */
    /* ALLOCATED:                                                           */
    /* FILLED OUT:                                                          */
    /* FREED:                                                               */

  struct saves save; /* Saves indicator of program stats                    */

  struct {           /* Data structures pointers, this is data that is not
                        required for the final computation                  */
    struct transithint *th;
    struct lineinfo    *li;
    struct atm_data    *at;
    struct extinction  *ex;
    struct opacity     *op;
    struct grid        *intens;
    struct optdepth    *tau;
    struct idxref      *ir;
    struct geometry    *sg;
#if 0
    struct savefiles   *sf;
#endif
    struct isotopes    *iso;
    struct molecules   *mol;
    struct outputray   *out;
    struct extcloud    *cl;
    struct extscat     *sc;
    struct detailout   *det;
    struct cia         *cia;
  }ds;
};
```

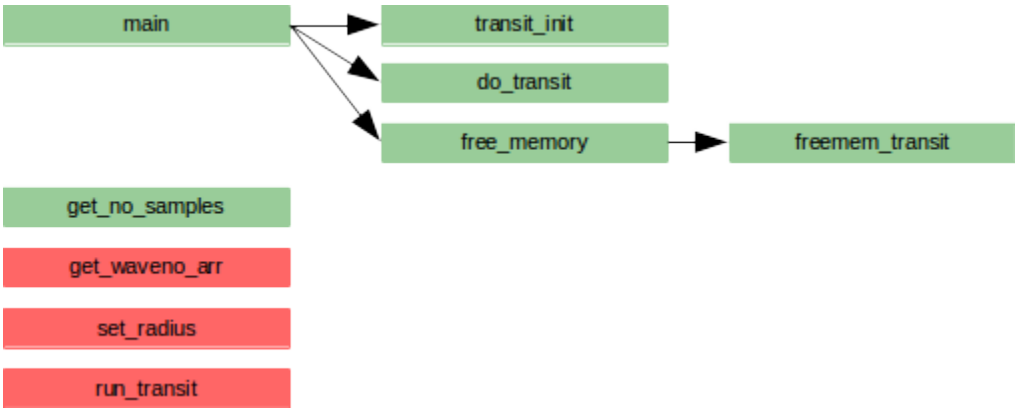

Figure 2: Function structure of transit.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, green functions are used in both, and red functions are unused at this time.

6 List of Functions in the `transit` Files

In this section we list all functions, sorted by source file, give a brief description of what they do, list all variables that they alter, and provide a walkthrough for each. A modified variable is written in `red`, a function is written in `blue`, and an unmodified variable is written in `typewriter`.

6.1 transit.c

This file contains the main `transit` driver functions. The `main` function calls `transit_init` and `do_transit`, the driver functions which do the initialization and calculation. Then `main` calls `free_memory` which frees the remaining memory. This structure is shown in Figure 2.

6.1.1 List of Functions Defined in transit.c

`void transit_init(int argc, char **argv)`
This function initializes the structures used in `transit`.

`int get_no_samples(void)`
Returns the size of the wavenumber sampling array.

`void get_waveno_arr(double *waveno_arr, int waveno)`
Fills the given array with the wavenumber sampling values.

`void set_radius(double refradius)`
Set the reference radius in the transit structure.

`void run_transit(double *re_input, int transitint, double *transit_out, int transit_out_size)`
Driver function that loads the atmospheric file and runs transit.

`void do_transit(double *transit_out)`
Driver function that calls all the functions to do calculations and then free the created arrays.

`void free_memory(void)`
Driver function that calls functions which free the rest of memory using in `transit`.

`int main(int argc, char **argv)`
Main driver function which calls functions to initialize, run, and then free `transit`.

`void freemem_transit(struct transit *tr)`
Free the transit structure

6.1.2 transit_init

Walkthrough
- Initialize the transit structure to 0.
- Call `processparameters` from argum.c to process the command line arguments and store them in the hint structure.
- Call `acceptgenhints` from argum.c to accept general hints from the hint structure.
- Call `printintro` from argum.c to print the introductory message.
- Turn off program warnings if verbosity is sufficiently low.
- Call `makewnsample` from makesample.c to create the wavenumber sampling.
- Call `getatm` from readatm.c to read the atmospheric file.
- Call `readlineinfo` from readlineinfo.c to read the TLI file.
- Call `makeradsample` from makesample.c to create the radius sampling.
- Call `opacity` from opacity.c to calculate the opacity grid.
- Call `readcia` from cia.c to read CIA file(s).
- Set boolean to indicate that `transit` has been initiated.

6.1.3 get_no_samples

Walkthough
- Return the size of the wavenumber array

6.1.4 get_waveno_arr

Walkthrough
- If `transit_init` has been run, fill out the given array with the wavenumber sampling values.
- Otherwise, indicate that `transit_init` has not been run and fill the given array with -1.

6.1.5 set_radius

Variables Modified
- Set `tr.r0` to the reference radius.

Walkthrough
- Set the reference radius in the transit structure.

6.1.6 run_transit

Walkthrough
- Call `realoadatm` from readatm.c to reload the atmospheric data.
- Call `do_transit` to run calculations.

6.1.7 do_transit

Variables Modified
- Fill in `tr.angleIndex`.
- Free `tr.save.ext`.

Walkthrough
- If `transit_init` has been run:
  - Call `makeipsample` from makesample.c to create the impact parameter sampling.
  - Call `interpolatecia` from cia.c to interpolate the CIA grid.
  - Call `idxrefrac` from idxrefraction.c to compute the index of refraction.
  - Call `extwn` from extinction.c to calculate the extinction coefficient.
  - Call `init_optdepth` to initialize optical depth structures.
  - If using eclipse geometry:
    - Call `tau` from tau.c to calculate optical depth as a function of radius.
    - Loop over each angle.
      - Fill in angle indices.
      - Call to `emergent_intens` from eclipse.c to calculate emergent intensity over the entire wavenumber range.
    - Call to `flux` from eclipse.c to calculate the flux spectrum.
    - Call to `freemem_intensityGrid` to free the intensity grid.
  - If using transit geometry:
    - Call to `tau` to calculate optical depth as a fnction of radius.
    - Call to `modulation` to calculate transit modulation at each wavenumber.
  - Free the saved extinction grid.
  - Call to `freemem_samp, freemem_idexrefrac, freemem_extinction, freemem_tau, freemem_outpu` to free the impact parameter sampling, index of refraction, extinction, optical depth, and output.
  - Increment the number of iterations.
- Otherwise warn that `transit_init` has not been run.

6.1.8 free_memory

Walkthrough
- Call to `freemem_molecules` to free molecular information.
- Call to `freemem_atmosphere` to free atmospheric data.
- If no opacity file was given (it was created), then call to `freemem_linetransition` to free the line transition data.
- Call to `freemem_lineinfo` to free line transition information.

- Call to `freemem_cia` to free CIA data.
- Call to `freemem_transit` to free the transit structure.
- Reset transit initiation boolean to 0.

6.1.9 main

Walkthrough
- Call to `transit_init` from transit.c to initialize the transit structures.
- Call to `get_no_samples` from transit.c to get the number of wavenumber samples.
- Call to `do_transit` from transit.c to run the main calculations.
- Call to `free_memory` to free all remaining allocated memory.
- Return success.

6.2 transitstd.c:

This file contains a number of standard functions. As they are largely unrelated, and called when necessary throughout the code, there is no general structure to the functions in this file.

6.2.1 List of Functions Defined in transitstd.c

`inline void transitdot(int thislevel, int verblevel, ...)`
Print a '.' character.

`int transiterror_fcn(int flags, const char *file, const long line, const char *str, ...)`
Set up additional arguments and call the error function.

`int vtransiterror_fcn(int flags, const char *file, const long line, const char *str, va_list ap)`
Error function for `transit`.

`int fileexistopen(char *in, FILE **fp)`
Check if a file exists and, if so, open it. Return various codes depending on what caused opening to fail.

`FILE * verbfileopen(char *in, char *desc)`
Handle various returns of `fileexistopen`. Returns the file pointer or raises an error.

`void transitcheckcalled(const long pi, const char *fcn, const int n, ...)`
Check that the 'n' functions (given as variable argument) have been called before 'fnc'. Raise an error if they have not.

`void error(int exitstatus, int something, const char *fmt, ...)`
Set up addition arguments and call the error function. Used for GSL errors.

`void freemem_molecules)(struct molecules *mol, long *pi)`
Free molecular information.

`void free_isov(prop_isov *isov)`
Free the partition function array in the variable isotope data structure.

`void free_isof(prop_isof *isof)`
Free the array (isotope name) in the fixed isotope information structure.

`void free_mol(prop_mol *molec)`
Free molecular data (density and abuncance arrays).

`void free_db(prop_db *db)`
Free the array (database name) in a database properties structure.

`void free_dbnoext(prop_dbnoext *dv)`
Free the temperatures array in a database properties structure.

`void free_samp(prop_samp *samp)`
Free the sampling values array in a sampling properties structure.

`void free_atm(prop_atm *atm)`
Free the pressure, temperature, and molecular mass arrays in the atmospheric properties structure.

`void savestr(FILE *out, char *str)`
Saves a string in binary to file.

`int reststr(FILE *in, char **str)`
Restores a string from a binary file.

`void linetoolong(int max, char *file, int line)`
Raise an error indicating that a line is too long.

`double timecheck(int verblevel, long iter, long index, char *str,`
`struct timeval tv, double t0)`
Print to screen the time since given time (t0).

6.2.2 transiterror_fcn:

Walkthrough
- Initialize variable arguments list.
- Call to `vtransiterror_fcn` from transitstd.c to print the error message and exit (unless indicated to continue).
- End using the variable arguments list.
- Return the return of `vtransiterror_fcn`.

6.2.3 vtransiterror_fcn:

Walkthrough

- Set up all the parts of the error message to be printed.
- If specified that no warnings should be given, return 0.
- Set the length of the error to the length of the first line of the error message.
- If the user has not specified that they want no error preamble, add the legnth of the error preamble and post-error message to the length of the total error message.
- If in debug mode, add the length of debug message to the error message length.
- Add the length of the specific error string to the error message length.
- Allocate memory for the error message and
- If the error preamble is desired, append it to the error message.
- Append the pre-error message to the error message.
- If in debug mode, print the debug message and append the debug message to the error message.
- If the error preamble is desired, append the error type to the error message.
- Append the specific error string to the error message.
- If the error preamble is desired, append the post-error message to the error message.
- Compose the error message string using the specifiec variable arguments and store in output.
- If an error occurred in composition, reallocate the size of the output and compose the error message again.
- Free the error message
- Print the output (the composed error message).
- Free the output.
- If the program is allowed to continue despite the error, increment the number of error encountered and return the size of the output.
- Exit program.

6.2.4 fileexistopen:

Walkthrough
- If a file was requested:
  - Check the status of the file. If an error occurs:
    - If it does not exist, return -1.
    - If a different error occurs, return -4.
  - If the file is not of a valid type (directory, device), return -2.
  - If the file is, for some other reason, unopenable, return -3.
  - If the file is successfully opened, return 1.
- If no file was requested, return 0.

6.2.5 verbfileopen:

Walkthrough
- Call to `fileexistopen` from transitstd.c to check if given file exists and open it if it does.
- If `fileexistopen` returns 1, return the file pointer.
- If `fileexistopen` returns 0, raise an error (no file given) and return NULL.
- If `fileexistopen` returns -1, raise an error (file does not exist) and return NULL.
- If `fileexistopen` returns -2, raise an error (file is invalid type) and return NULL.
- If `fileexistopen` returns -3, raise an error (file is unopenable, likely due to permissions) and return NULL.

- If `fileexistopen` returns -4, raise an error (file exists, but something else wrong) and return NULL.
- Otherwise, raise an error.
- Return NULL.

6.2.6 transitcheckcalled:

Walkthrough
- Check each given function against the progress indicator.
- If one of the functions has not been called, append the error message.
- Append the names of the functions not called.
- Call `transiterror` to print the error.

6.2.7 error:

Walkthrough
- Create output string.
- Call `vtransiterror_fcn` from transitstd.c to print the error.
- Exit the program.

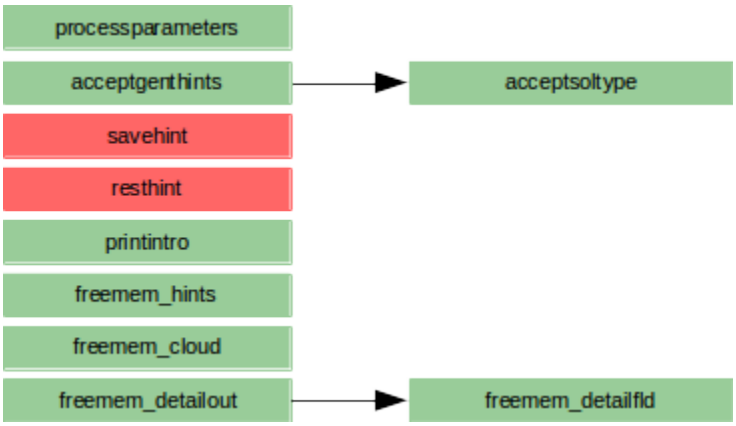

Figure 3: Function structure of argum.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, green functions are used in both, and red functions are unused at this time.

6.3 argum.c:

Functions in this file handle the creation and filling-out of the hint structure. The hinted values are taken from command-line arguments and/or a configuration file, and then the general hinted values are placed into the main transit structure. `savehint, resthint` are unused. Figure 3 shows the function structure.

6.3.1 List of Functions Defined in argum.c:

`int processparameters(int argc, char **argv, struct transit *tr)`
Generate the command-line option parser. Initialize transithint and populate it's variables based on the command-line arguments.

`int acceptsoltype(transit_ray_solution **sol, char *hname)`
Initialize transit ray solution sol. and determine if any of sol-¿name matches hname.

`int acceptgenhints(struct transit *tr)`
Set output file names in transit (out, toomuch, and sample). Initialize transit.sol. Set geometry and detailed output variables in transit.

`void savehint(FILE *out, struct transithint *hints)`
Saves hints structure.

`int resthint(FILE *in, struct transithint *hint)`
Restore hints structure. The structure needs to have been allocated before.

`void printintro()`
Print the introductory message.

`void freemem_hints(struct transithint *h)`

Frees hints structure.

`void freemem_cloud(struct extcloud *c)`
Free cloud structure. This function is intended to be used when cloud functionality is added but is not used currently.

`void freemem_detailout(struct detailout *d)`
Driver function to free stored extinction, optical depth, and CIA data.

`void freemem_detailfld(struct detailfld *f)`
Free a single field of data stored in detailout structure.

6.3.2 processparameters:

Variables Modified:
- Set `tr.ds.th.verbnoise, tr.ds.th.mass, tr.ds.th.savefiles` (verbosity, abuncance type, and file saving boolean) to defaults.
- Set `tr.ds.th.ncia, tr.ds.th.ciafile` (number of CIA files, CIA filenames) from command line arguments/config file.
- Set `tr.ds.th.save.ext` (saved extinction grid) from command line arguments/config file.
- Set `tr.ds.th.f_opa` (opacity filename) from command line arguments/config file.
- Initialize `tr.ds.th.det.cia, tr.ds.th.det.tau, tr.ds.th.det.ext` (detailed field structures for CIA, optical depth, and/or extinction) if specifed by command line arguments/-config file.
- Fill in the `tr.ds.th.det` structure values.
- Set `tr.ds.th.ethresh` (factor threshold for line profile calculation) from command line arguments/config file.
- Allocate and set `tr.ds.th.solname` (ray solution type) from command line arguments/-config file.
- Allocate and set `tr.ds.th.f_atm, tr.ds.th.f_line, tr.ds.th.f_molfile, tr.ds.th.f_outmod, tr.ds.th.f_outsample, tr.ds.th.f_toomuch, tr.ds.th.f_outflux, tr.ds.th.f_outintens` (input and output filenames) from command line arguments/config file.
- Set `tr.ds.th.savefiles` (boolean to save files or not) from command line arguments/-config file.
- Set `tr.ds.th.p0, tr.ds.th.r0, tr.ds.th.gsurf` (surface pressure, radius, and gravity) from command line arguments/config file.
- Set `tr.ds.th.allowrq` (variance from unity allowed in total abundance) from command line arguments/config file.
- Set `tr.ds.th.qmol, tr.ds.th.qscale` (species with scale factors, scale factors) from command line arguments/config file.
- Set `tr.ds.th.opabreak` (boolean to indicate to the program to break after opacity calculation) from command line arguments/config file.
- Set `tr.ds.th.rads.i, tr.ds.th.rads.f, tr.ds.th.rads.d, tr.ds.th.rads.fct` (radius sampling initial value, final value, spacing, and units conversion factor) from command line arguments/config file.

- Set tr.ds.th.wavs.i, tr.ds.th.wavs.f, tr.ds.th.wavs.d, tr.ds.th.wavs.fct (wavelength sampling initial value, final value, spacing, and units conversion factor) from command line arguments/config file.
- Set tr.ds.th.wns.i, tr.ds.th.wns.f, tr.ds.th.wns.d, tr.ds.th.wns.fct, tr.ds.th.wns.o (wavenumber sampling initial value, final value, spacing, units conversion factor, and oversampling factor) from command line arguments/config file. Initialize tr.ds.th.wns.n (number of samples) to 0 and tr.ds.wns.v (sampling values) to NULL.
- Set tr.ds.th.temp.i, tr.ds.th.temp.f, tr.ds.th.temp.d (temperature sampling initial value, final value, and spacing) from command line arguments/config file.
- Set tr.ds.th.timesalpha (number of half-widths in a Voigt profile) from command line arguments/config file.
- Set tr.ds.th.nDop, tr.ds.th.nLor (number of Doppler and Lorentz broadening width samples) from command line arguments/config file.
- Set tr.ds.th.dmin, tr.ds.th.dmax, tr.ds.th.lmin, tr.ds.th.lmax (broadening width sample boundaries) from command line arguments/config file.
- Set verbosity to 0 if the 'q' command-line argument is given.
- Set verbosity to specified value if the 'v' command-line argument is given.
- Set tr.ds.th.sg.starrad (star radius) from command line arguments/config file.
- Set tr.ds.th.sg.smaxis, tr.ds.th.sg.time, tr.ds.th.sg.incl, tr.ds.th.sg.ecc, tr.ds.th.sg.lnode, tr.ds.th.sg.aper (semimajor axis, phase from eclipse, inclination, eccentricity, longitude of ascending node, argument of pericenter) from command line arguments/config file.
- Set tr.ds.th.sg.smaxisfct, tr.ds.th.sg.timefct, tr.ds.th.sg.inclfct, tr.ds.th.sg.eccfct, tr,.ds.th.sg.lnodefct, tr.ds.th.sg.aperfct (unit conversion factors for above orbital parameters) from command line arguments/config file.
- Set tr.ds.th.sg.transpplanet (boolean to set maximum optical depth to toomuch) to True if specified in commmand-line arguments/config file.
- Set tr.ds.th.toomuch (maximum optical depth to make calculations, above which it is assumed no light transmits) from command line arguments/config file.
- Set tr.ds.th.taulevel (optical depth integration level) from command line arguments/-config file.
- Set tr.ds.th.modlevel (modulation integration level) from command line arguments/-config file.
- Set tr.ds.th.cl.cloudext, tr.ds.th.cl.cloudtop, tr.ds.th.cl.cloudbot (maximum cloud extinction, initial cloud radius, and final cloud radius) from command line arguments/config file.
- Set tr.ds.th.angles (intensity angles) from command line arguments/config file.

Walkthrough:

- Set up an enumerated list of all command line arguments, creating a key.
- Build a structure to identify all command line arguments.
- Build a configuration paramters structure.
- Initialize the hint structure, and set all memory to zero.
- Set up flags, verbosity, abundance units, and whether or not to save files.
- Set up the detailed output field structures for optical depth, extinction, and CIA.
- Begin infinite loop:
  - Call to procopt from procopt.c to process the command line options. If the option

supplies a configuration file, procopt will parse the file.
  - If procopt returns -1 (indicating no more command line arguments to process), exit the loop.
  - Handle the returns of procopt on a case-by-case basis (a case for each command line argument), filling in the hint structure with the specified values. There are too many options to practically list here. makes the command-line-argument parser, resets the transithint struct, and fill in its variables with default values and command line arguments.
- Call procopt_free from procopt.c to free the memory used by procopt.
- Return 0 on success.

6.3.3 acceptsoltype:

Walkthrough
- Loop over each element in the ray solutions array.
  - Compare each solution with the given string.
  - If they match, set the solution and return 0.
- Return -1.

6.3.4 acceptgenhints:

Variables Modified
- Copy tr.f_outmod from th.f_outmod or default (modulation output filename).
- Copy tr.f_outflux from th_outflux or default (flux output filename).
- Copy tr.f_toomuch, tr.f_outsample, tr.outintens from th.f_toomuch, th.f_outsample, th.f_outintens (maximum optical depth, sampling output, and intensity output filenames).
- Copy tr.ds.det from th.det (detailed output structure).
- Copy tr.timesalpha from th.timesalpha (Voigt profile width).
- Copy tr.opabreak from th.opabreak.
- Set tr.interpflag to SAMP_LINEAR or SAMP_SPLINE depending on tr.fl (flag).
- Copy tr.r0 from th.r0 (reference radius).
- Copy tr.p0 from th.p0 (reference radius).
- Copy tr.gsurf from th.gsurf (surface gravity).
- Call parseArray from iomisc.c to copy tr.qscale from th.qscale and set tr.nqmol to the size of th.qscale. If th.qscale was not given, set tr.nqmol to 0.

Walkthrough:
- Set output filenames for modulation, flux, radius where maximum tau was reached, sampling, and intensity from hint structure.
- Set molecular filename from hint structure.
- Call to acceptsoltype from argum.c to get the solution type. Raise an error if an invalid type was provided, and exit program.
- Call to setgeomhint from geometry.c to set hinted geometry information.
- Copy the hinted detailed output structure.
- Check that the given number of alpha units in Voigt profile width is more than 1. If not, raise an error and return -1.
- Set the number of alpha units in Voigt profile width.

- Check that the transition line strength threshold is positive. If not, raise an error and return -1.
- Call `transitacceptflag` (defined in transit.h) to pass atmospheric flags into the transit structure.
- Set the flag to break `transit` after the opacity grid has been calculated from the hint structure.
- Set the interpolation function flag. Raise an error if invalid function specified.
- Raise an error and return -1 if the specified reference radius is negative.
- Set the reference radius from the hint structure.
- Raise an error and return -1 if the specified reference pressure is negative.
- Set the reference pressure from the hint structure.
- Raise an error and return -1 if the specified surface gravity is negative.
- Set the surface gravity from the hint structure.
- If abundance scale factors were specified:
  - Call `parseArray` from iomisc.c to set the abundance scale factors from the hint structure and set the number of scale factors.
  - If the number of molecules with scale factors does not match the size of the scale factors array, raise an error.
- Otherwise, set the number of scale factors to 0.
- Return 0 on success.

6.3.5 savehint:

Walkthrough

- Write the hint structure to file.
- Call to `savestr` from transitstd.c to write input and output filenames, in binary, to file (atmosphere, TLI, CIA, modulation, flux, intensity, radius where maximum tau was reached, and sampling files). Write the solution name to file.
- Call to `savesample_arr` from makesample.c to save the radius, wavelength, wavenumber, and impact parameter sampling to file.

6.3.6 resthint:

(FINDME: do we doc this one?)

6.3.7 freemem_hints:

Variables Modified

- Free `th.f_atm, th.f_line, th.f_outmod, th.f_outflux, th.f_outintens, th.f_toomuch, th.f_outsample, th.f_molfile, th.solname, th.ciafile`.

Walkthrough

- Free all filenames in the hint structure.
- Free the solution name.
- Call to `freemem_samp` from makesample.c to free the hinted sampling for radius, wavelength, wavenumber, and impact parameter.
- Call to `freemem_cloud` from argum.c to free hinted cloud info.
- Call to `freemem_detailout` from argum.c to free hinted detailed output structure.

6.4 geometry.c: EDITED

This file contains routines which handle the hinted geometry parameters and make calculations regarding the geometry of the transit or eclipse.

6.4.1 List of Functions Defined in geometry.c:

`int setgeomhint(struct transit *tr)`
Set transit geometry variables (tr.ds.sg) from hint or default variables.

`int setgeom(struct geometry *sg, double time, long *flags)`
Set x and y geometry variables (coordinates of the planet relative to the star).

`inline PREC_RES starvariation(double x, double y, double radius)`
Evaluate if $x^2 + y^2 >$ radius^2. Return 0 if so. Otherwise return 1.

6.4.2 setgeomhint

Modified
- Copy `th.sg.transpplanet` into `tr.ds.sg.transpplanet`.
- Set all `tr.ds.sg` variables except `tr.ds.sg.x` and `tr.ds.sg.y` from `th.sg`. If a hinted value is not given, set them to default values.
- Update `tr.pi` to account for `TRPI_GEOMETRYHINT`.

Walkthrough
- Copy hinted transpplanet into the transit structure. This is a boolean which, if true, sets the maximum optical depth to toomuch.
- Set all variables in the geometry structure from the hinted structure (`tr.ds.th.sg`) except X and Y values (coordinates of planet relative to the star). If no hinted variable is given, the variables are set to a default value.
- Update the progress indicator to account for `TRPI_GEOMETRYHINT`.
- Return 0 on success.

6.4.3 setgeom

Modified
- Calculate `sg.x, sg.y` by solving the Kepler equation.
- Update `tr.pi` to account for `TRPI_GEOMETRY`.

Walkthrough
- Calculate semi-major axis, eccentricity, inclination, observation time, and stellar mass in cgs units.
- Set the precision limit for the square of the eccentric anomaly.
- Calculate mean motion (orbital angular frequency).
- Set the approximate eccentric anomaly and calculate the eccentric anomaly
- While the square of the difference between the eccentric anomaly approximation and the eccentric anomaly is greater than the precision limit, set the approximation equal to the

eccentric anomaly and recalculate the eccentric anomaly at time t. When the loops exits, the eccentric anomaly at time t will have been calculated.
- Calculate orbital parameters.
- Calculate the position of the planet relative to the center of the star (`sg.x, sg.y`).
- Update the progress indicator to account for `TRPI_GEOMETRY`.
- Return 0 on success.

6.4.4 starvariation

Walkthrough
- Return 0 if position (x, y) is not within a circle of given radius.
- Return 1 otherwise.

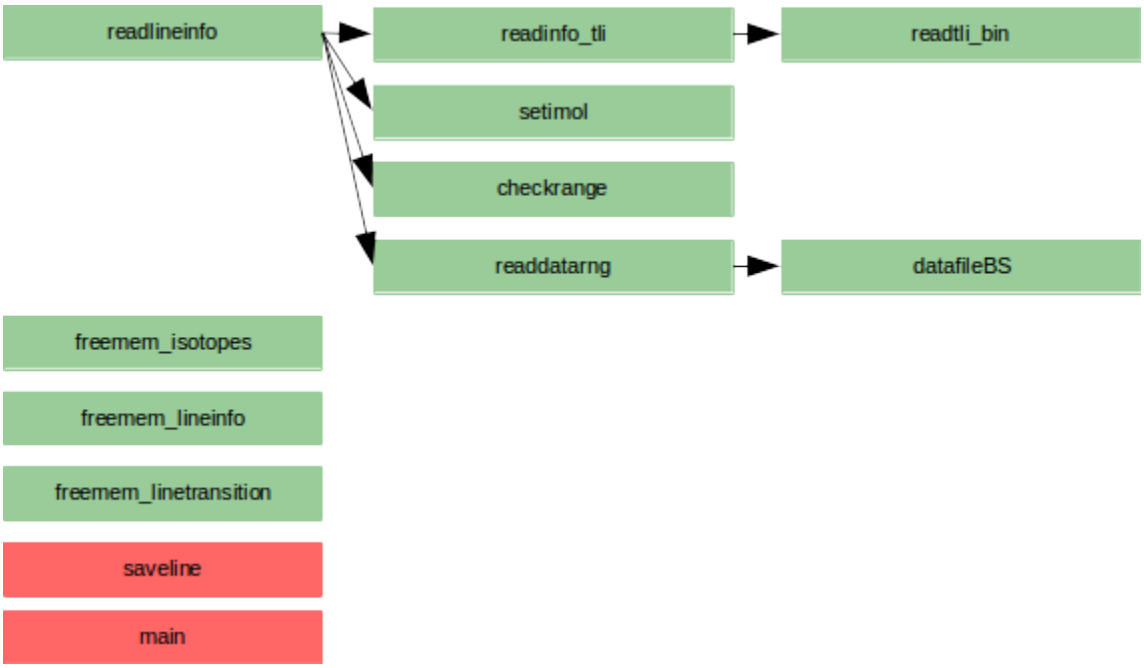

Figure 4: Function structure of readlineinfo.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, green functions are used in both, and red functions are unused at this time.

6.5 readlineinfo.c:

This file is concerned with reading TLI files produced by the `pylineread` program. Only binary TLI files are accepted. Functions `saveline, main` are not currently used in `transit`. Figure 4 shows this function structure.

6.5.1 List of Functions Defined in readlineinfo.c:

```
static inline void datafileBS(FILE *fp, PREC_NREC offs, PREC_LNDATA target,
                              PREC_NREC *resultp, int reclength, int up)
```

Do a binary search in file pointed by 'fp' between 'off' and 'off+nfields' looking for 'target' as the first item of a record of length 'reclength', result index (with respect to offs) is stored in 'resultp'.

```
int readlineinfo(struct transit *tr)
```

Driver function to read TLI: read isotopes info, check margin and ranges, and read line transition information.

```
int readinfo_tli(struct transit *tr, struct lineinfo *li)
```

Check if a TLI file exists. Check that machine formating is compatible with lineread. Determine if TLI is ASCII or binary. Read either ASCII or binary TLI file. Declare line_transition.

```
int readtli_bin(FILE *fp, struct transit *tr, struct lineinfo *li)
```

Read initial and final wavelength limits and number of databases. Allocate pointers to

database, and isotope arrays. Get databases info: names, number of temperatures, temperatures, number of isotopes, isotope names and masses, partition function, and cross sections. Get cumulative number of isotopes.

`int setimol(struct transit *tr)`
Set each isotope's molecular identifier number.

`int checkrange(struct transit *tr, struct lineinfo *li)`
Initialize wavelength sample struct. Set margin. Set initial and final wavelengths to use. Check that margin leaves a non-zero wavelength range.

`int readdatarng(struct transit *tr, struct lineinfo *li)`
Read and store the line transition info (central wavelength, isotope ID, lowE, log(gf)) into lineinfo. Return the number of lines read.

`int freemem_isotopes(struct isotopes *iso, long *pi)`
Frees isotope structure.

`int freemem_linetransition(struct line_transition *lt, long *pi)`
Frees line transition data.

`int freemem_lineinfo(struct lineinfo *li, long *pi)`
Frees line transition info.

`void saveline(FILE *fp, struct lineinfo *li)`
Saves line information.

`int main(int argc, char **argv)`
For debugging only.

6.5.2 readlineinfo

Variables Modified
- Reset `tr.ds.li, tr.ds.iso` (lineinfo and isotopes structures).
- If an opacity file exists, update `tr.pi` to account for TRPI_READINFO and TRPI_READDATA.

Walkthrough:
- Reset line information and isotopes structures.
- Call `readinfo_tli` to check if TLI file exists, open it, and get header information (all info exept line transitions).
- Call `setimol` to set the molecular index of each isotope.
- Call `checkrange` to set up the lineinfo wavelength sampling and wavelength margin.
- Check if an opacity file exists. If not, call `readdatarng` to read the TLI file data. Otherwise, skip reading the TLI file and update the progress indicator to allow the program to continue.
- Return 0 on success.

6.5.3 readinfo_tli

Variables Modified
- Set `tr.f_line` from `th.f_line` if file exists and could be opened (TLI filename).
- Set `tr.fp_line` (pointer to TLI file).
- Set `tr.ds.li.asciiline` according to the format of the TLI file.
- Declare `tr.ds.li.lt`.
- Set `tr.ds.li.lt.wfct, tr.ds.li.lt.efct` from default values (line-transition wavelength and lowE units factor).
- Update `tr.pi` to include `TRPI_READINFO`.

Walkthrough:
- Declare a union variable which is used to determine the type of the TLI file (ASCII or binary) and to check endianness compatibility.
- Check that a TLI file name was given. If not, raise an error and return -2.
- Check if TLI file exists. If not, raise an error and return -1.
- Set the TLI file name and file pointer from hint structure.
- Read the first four bytes of the TLI file into the union variable.
- If the file appears to be an ASCII file:
  - If the first four bytes of the TLI file indicate the file is ASCII, compare them with '#TLI'. If they match, read the next six bytes, append them to the first four and compare with '#TLI-ascii'. If either compare did not match, raise an error and return -3.
  - Set the ASCII file indicator to 1.
  - Read past the rest of the first line of the TLI file.
- Call to `readtli_bin` to read the binary TLI file. Raise an error and return -6 if `readtli_bin` returns an error.
- Set the wavelength and lowE units conversion factor for line transitions.
- Update the progress indicator.
- Return -1 on success.

6.5.4 readtli_bin:

Variables Modified
- Set `tr.ds.li.tli_ver, tr.ds.li.lr_ver, tr.ds.li.lr_rev` from TLI file (lineinfo TLI version, lineinfo version and revision).
- Allocate `tr.ds.iso.db` (database structures for each isotope).
- Allocate `tr.ds.li.db` (database structures for temperature information).
- Allocate `tr.ds.iso.isof` (structure for fixed isotope information).
- Allocate `tr.ds.li.isov` (structure for variable isotope information).
- Allocate `tr.ds.iso.isoratio` (isotope abundance ratio).
- Allocate `tr.ds.iso.db.n, tr.ds.iso.db.molname` (database name and molecule name) for each database.
- Set `tr.ds.li.db.t, tr.ds.iso.db.i` (number of temperatures and number of isotopes) for each database.
- Allocate `tr.ds.li.db.T` (temperature points in TLI file) for each database and set from the TLI file.
- Reallocate `tr.ds.li.isov, tr.ds.iso.isof, tr.ds.iso.isoration` to account for new isotopes.

- Allocate tr.ds.li.isov.z (partition function).
- Set tr.ds.iso.isof.d (database index of the isotope) for each isotope.
- Allocate and set tr.ds.iso.isof.n (isotope name) for each isotope.
- Set tr.ds.iso.isof.m (mass) for each isotope.
- Set tr.ds.iso.isoratio (isotopic ratio) for each isotope.
- Set tr.ds.li.isov.z (partition function) for each isotope.
- Set tr.ds.li.isov.n (partition function array length) for each isotope.
- Set tr.ds.iso.db.s (index of the first isotope) for each database (species).
- Set tr.ds.li.ni, tr.ds.iso.n_i (number of isotopes).
- Set tr.ds.li.ndb, tr.ds.iso.n_db (number of databases).
- Set tr.ds.li.iniw, tr.ds.li.finw (initial and final wavelength).
- Set tr.ds.li.endinfo (position of beginning of transition data).
- Allocate tr.ds.iso.isov (structures for isotopes' variable data).

Walkthrough

- Read the TLI version, lineread version, and lineread revision number from the TLI file.
- Check that the TLI version is compatible with the transit version. If not, raise an error.
- Read the initial wavelength, final wavelength, and number of databases from the TLI file.
- Allocate structures for databases for each isotope.
- Allocate structures for fixed and variable isotope data.
- Allocate isotopic abundance ratios.
- Loop over each database (each species):
  - Read database name length, allocate space for the name, and read the name from the TLI file.
  - Read molecule name length, allocate space for the name, and read the name from the TLI file.
  - Read and set the number of temperatures and number of isotopes.
  - Allocate array for the temperatures and read from the file.
  - Reallocate variable isotope data structures, fixed isotope data structures, and isotopic abundance ratio to account for new isotopes.
  - Allocate array for partition function data.
  - Loop over each isotope:
    - Set isotope's database index number.
    - Read isotope name length, allocate space for the name, and read the name from the TLI file.
    - Read isotope mass and isotopic ratio from the TLI file.
  - Set the index of the first isotope in this isotope.
  - Increment the number of isotopes by the number of isotopes in this database.
- Set the number of total isotopes, number of databases, position of the first transition in the TLI file, initial wavelength, final wavelength, and number of databases (in both the line transition and isotopes structures).
- Allocate structures for isotopes' variables data.
- Return 0 on success.

6.5.5 checkrange:

Variables Modified:

- Set tr.ds.li.wavs.fct from tr.ds.th.wavs.fct if provided, else default to cgs units (lineinfo wavelength units factor).
- Set tr.ds.li.wavs.f from th.wavs.f if provided, else default to maximum wavelength in database tr.ds.li.wf*TLI_WAV_UNITS/tr.ds.li.wavs.fct (lineinfo final wavelength).
- Set tr.ds.li.wavs.i from th.wavs.i if provided, else default to minimum wavelength in database tr.ds.li.wi*TLI_WAV_UNITS/tr.ds.li.wavs.fct (lineinfo initial wavelength).
- Update tr.pi to account for TRPI_CHKRNG.

Walkthrough:
- Initialize lineinfo's wavelength sample structure.
- Raise an error if the user supplied a negative wavelength factor.
- Set the lineinfo wavelength unit conversion factor from the transithint structure
- Check that the user supplied a positive final wavelength. If not, raise a warning and set to the default (maximum database wavelength).
- If the user supplied a valid final wavelength:
  - Check that the final wavelength is not lower than the initial wavelngth. If so, raise an error and return -3.
  - Check that the final wavelength is not higher than the maximum database wavelength. If so, raise a warning.
  - Set the final wavelength from the hint structure.
- Check that the user supplied a positive inital wavelength. If not, raise a warning and set to the default (minimum database wavelength).
- If the user supplied a valid wavelength:
  - Check that the initial wavelength is not higher than the maximum database wavelength. If so, raise and error and return -2.
  - Check that the initial wavelength is not smaller than the minimum database wavelength. If so, raise a warning.
  - Set the initial wavelength from the hint structure.
- Update the progress indicator.

6.5.6 readdatarng:

Variables Modified
- Allocate tr.ds.li.lt.wl, tr.ds.li.lt.isoid, tr.ds.li.lt.gf, tr.ds.li.lt.elow (line-transition's wavelength, isotope ID, gf, and lower state energy).
- Set tr.ds.li.lt.wl, tr.ds.li.lt.isoid, tr.ds.li.lt.gf, tr.ds.li.lt.elow from read TLI values.
- Set tr.ds.li.n_l (Number of lines read from TLI).
- Update tr.pi to include TRPI_READDATA.

Walkthrough
- Call to fileexistopen from iomisc.c to open the TLI file. Return -1 if the file cannot be opened.
- Check if the file is 'seekable'. If not, raise an error and return -2.
- Move the file pointer to the beginning of the transition data.
- Read the number of transitions from the TLI file.
- Read the number of isotopes from the TLI file.

- Read the number of transitions per isotope from the TLI file.
- Loop over subsequent wavelength entries to check that they are greater than the final wavelength. If not, increment the index of the final wavelength until the condition is true.
- Store the number of lines.
- Allocate arrays for line transition's oscillator strength, central wavelength, isotope ID, and lower-state energy.
- Check for allocation errors. Raise an error if any of the allocations failed.
- Set the starting location for wavlengths, isotope IDs, lower-state energy, and oscillator strength.
- Loop over each isotope:
  - Call to `datafileBS` to find the index of the first transition to be read.
  - Call to `datafileBS` to find the index of the last transition to be read.
  - Move file pointer to the beginning of the wavelength info and read into the allocated array. Do the same for isotope IDs, lower-state energy, and oscillator strength (gf).
  - Increment the number of lines read.
  - Move the wavelength offset to the next isotope.
- Reallocate the central wavelength, isotope ID, lower-state energy, and osciallator strength arrays to the correct size.
- Close the file.
- Update progress indicator.
- Return the number of lines read.

6.5.7 datafileBS:

Walkthrough

- Set the index of the end of the search range to one less than the number of fields to search. Set the index of the beginning of the range to 0.
- Perform binary search. While the difference between the beginning and end indices is greater than 1:
  - Set the result index to the middle of the search range.
  - Move the file pointer to the result index.
  - Read the value at that point.
  - If the target value is greater than the read value, move the beginning of the search range up to the result index. Otherwise, move then end of the search range to the result index.
- Perform a linear search through entries above or below that found by the binary search depending on the flag passed.
- Set the result index to the beginning index of the search range.
- Move the file pointer to this point.
- Read the value at that point in the file.

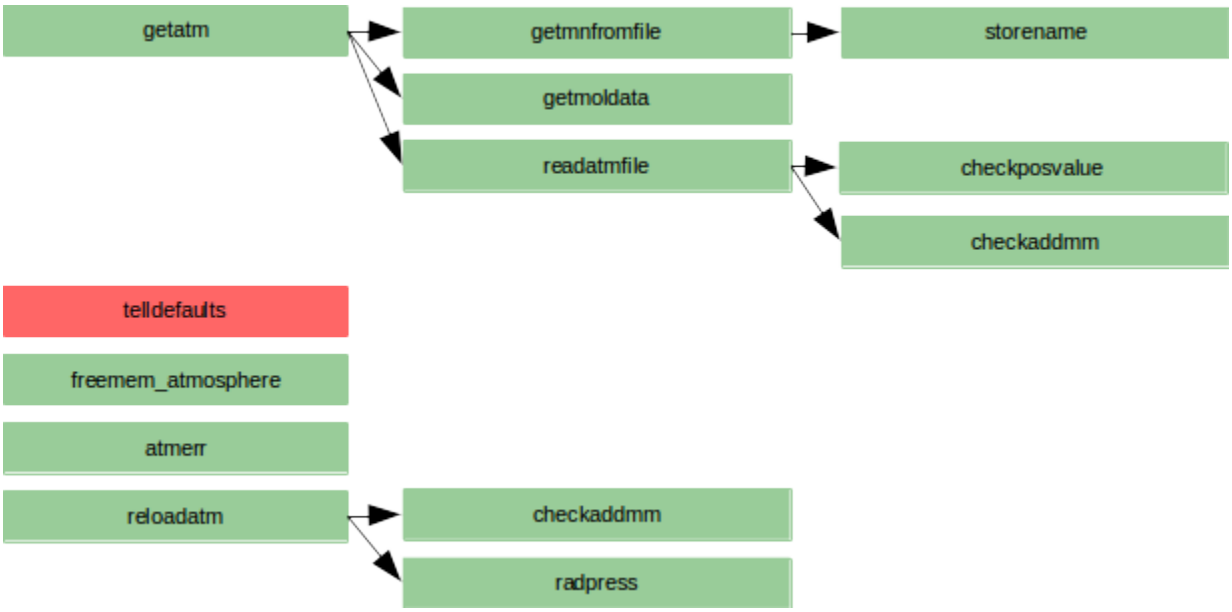

Figure 5: Function structure of readatm.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, green functions are used in both, and red functions are unused at this time.

6.6 readatm.c:

This file contains functions which read the atmospheric file. `telldefaults` is currently unused. Figure 5 shows the function structure.

6.6.1 List of Functions Defined in readatm.c:

```
int getatm(struct transit *tr)
```
Initialize ds.at (atm_data). Set abundance mass and allowrq parameters. Check existence, open, and set pointer to atmosphere file. Get keyword variables from atm file (list of isotopes among others). Get temperature and isotopes abundances per radius from atm file.

```
double checkaddmm(double *mm, PREC_NREC r, prop_isov *isov, prop_isof *isof,
                  int n, _Bool mass, enum isodo *isodo)
```
Compute the mean molecular mass, check that sum of abundances is no bigger than 1, and return it.

```
void telldefaults(struct isotopes *iso, struct atm_data *at)
```
Tell defaults when only one radius is being selected.

```
int freemem_atmosphere(struct atm_data *at, long *pi)
```
Free memory from the atmosphere structure.

```
void storename(struct atm_data *at, char *line)
```
Store info about the atmosphere file.

`static void atmerr(int max, char *file, int line)`
Print error message when a line of the file is longer than the max characters.

`static void invalidfield(char *line, int nmb, int fld, char *fldn)`
Print an error message when a field with transition info is invalid.

`static inline void checkposvalue(PREC_RES val, int field, long line)`
Chack that a value is positive, and raise an error if it is not.

`int getmnfromfile(FILE *fp, struct atm_data *at, struct transit *tr)`
Get keyword variables from atmosphere file (mass/number abundance bool; zero-radius offset; radius, temperature, and pressure units factor; atmfile name/info; list isotopes; list of proportional-abundance isotopes). Store molecules and proportional isotopes in atm_data struct. Determine which linedb isotope corresponds to such atm_data isotope. Solve non-matched linedb isotope cases. Put all non-ignore isotopes in transit.ds.iso structure.

`int readatmfile(FILE *fp, struct transit *tr, struct atm_data *at,`
`prop_samp *rads, int nrad)`
Read and store radius, pressure, and temperature from file. Read abundances for each (non other-factor) isotope. Sum fractional abundances. Calculate ramaining (other-factor) abundances. Calculate mean molecular mass per radius. Calculate densities per isotope at each radius.

`void getmoldata(struct atm_data *at, struct molecules *mol, char *filename)`
Read and store non-layer-dependent molecular data (mass, radius, ID) and store in mol struct.

`int reloadatm(struct transit *tr, double *input)`
Reload data from array into transit's atm structure.

`int radpress(double g, double p0, double r0, double *temp, double *mu,`
`double *pressure, double *radius, intnlayer, double rfct)`
Recalculate radius array according to hydrostatic pressure, and find the radial location of the reference pressure.

6.6.2 getatm:

Variables Modified:
- Initialize `tr.ds.at, tr.ds.mol` (atm_data, molecules).
- Set `tr.ds.at.mass` from `th.mass` (mass or number abundance bool).
- Set `tr.allowrq` from `th.allowrq` (minimum allowed sum of abundances).
- Set `tr.fp_atm` from `th.f_atm` if `th.f_atm` exists and can be opened (atmosphere file pointer).
- Set `tr.f_atm` (atmosphere file name).
- Set `tr.ds.at.atm.tfct, tr.ds.at.atm.pfct` from default values (temperature and pressure unit factors).

- Allocate `tr.ds.at.rads.v, tr.ds.at.atm.t, tr.ds.at.atm.p` (radius, temperature and pressure array).
- Allocate `tr.ds.mol.nmol, tr.ds.mol.ID, tr.ds.mol.mass, tr.ds.mol.radius, tr.ds.mol.molec` (number of molecules, molecular IDs, molecular masses, molecular radii, and molecular properties).
- Allocate `tr.ds.at.molec, tr.ds.at.mm, tr.ds.at.molec.d, tr.ds.at.molec.q` (molecular properties substructure, mean molecular mass, molecular density, molecular abundance) and set `tr.ds.at.molec.n` (size of radius sampling)
- Set `tr.ds.at.rads.i, tr.ds.at.rads.f, tr.ds.at.rads.o, tr.ds.at.rads.d` (radius sampling initial value, final value, oversampling, and spacing).
- Update `tr.pi` to account for `TRPI_GETATM`.

Walkthrough:
- Initialize atmosphere and molecular structures by setting memory to 0.
- Copy mass boolean from the transithint structure. This boolean indicates whether abundances are in units mass or number.
- Copy abundance exactness number from transithint structure. This number determines if an error is raised when the sum of the abundances does not equal one.
- If the atmospheric file was not specified, raise an error and return -1.
- If the atmospheric file is specified, exists, and can be opened, set the file pointer and file name.
- Allocate radius sampling values, temperature, and pressure arrays.
- Call `getmnfromfile` from readatm.c to get keyword variables from the atmospheric file. Raise an error if the atmospheric file contains less than 1 line read.
- Allocate molecular structure variables (number of molecules, molecular ID, molecular masses, molecular radii, and molecular properties substructure).
- Call `getmoldata` to get molecular data from the molecules file.
- Allocate molecular properties substructure of atmospheric data structure, mean molecular mass, molecular density, and molecular density. Set number of elements.
- Call to `readatmfile` to read per-radius isotopic abundances, temperatures and set number a radius layers.
- Close the file.
- Set the radius sampling initial value, final value, oversampling, and spacing.
- Update progress indicator to show `getatm` has been run.
- Return 0 on success.

6.6.3 checkaddmm:

Walkthrough
- Raise an error if given radius layer is beyond the allocated radius layers.
- Compute mean molecular mass and sum of abundances.
- If the sum of abundances is more than 0.1% over unity, raise a warning.
- Return the sum of the abundances.

6.6.4 getmnfromfile:

Variables Modified
- Set `tr.ds.at.begline` (line where radius-dependent info begins) to 0.

- Allocate and fill out `tr.ds.mol.name` (molecules' names).
- Set `tr.ds.at.mass` according to the atmospheric file.
- Set `tr.ds.at.info` according to the atmospheric file.
- Set `tr.ds.at.begpos` (position of the beginning of data).

6.6.5 readatmfile:

Variables Modified

- Reallocate `tr.ds.rads.v, tr.ds.at.atm.t, tr.ds.at.atm.p, tr.ds.at.mm, tr.ds.at.molec.d, tr.ds.at.molec.q` (radius sampling values, temperature, pressure, mean molecular mass, density, and abundance arrays) to accommodate more radius layers.
- Set `tr.ds.at.molec.n` to the new number of radius layers.
- Fill in `tr.ds.rads.v, tr.ds.at.atm.p, tr.ds.at.atm.t` from atmosphere file.
- Fill in `tr.ds.at.molec.q` from atmosphere file.
- Calculate `tr.ds.at.molec.d`.

Walkthrough

- Call `valueinarray` to find the indices of H2 and He in the molecular ID array.
- Move the stream position to the beginning of data in the atmosphere file.
- Call `fgetupto_err` to read past all blank lines and comments.
- Call `countfields` to count the number of values per line, minus the radius, pressure, and temperature columns.
- Move the stream position back to the beginning of data in the atmosphere file.
- Begin infinite loop.
  - If the current radius index reaches the total number of radius layers:
    - Perform a binary left shift to double the number of radius layers.
    - Reallocate radius sampling values, temperature, pressure, mean molecular mass, density, and abundance arrays according to the new number of radius layers.
    - Set the number of radius elements in the molecules' substructures to the new number of radius layers.
  - Call `fgetupto_err` to skip past comments and blank lines.
  - Break loop when the end of the file is reached.
  - Store radius values in the radius sampling values array. Call `checkposvalue` from readatm.c to check that the stored value is positive
  - If there was a problem converting the read value to a double, call `invalidfield` from readatm.c to warn that an invalid value was given in the file.
  - Loop over each abundance.
    - Read the abundance for this particular isotope and radius into the corresponding molecular abundance array.
    - Convert abundances using the scale factor.
    - Sum up abundances and metal abundances (everything but H2 and He).
    - Check that the abundances are positive, and raise an error if there was a problem reading the abundance into the array.
  - Calculate H2/He ratio, Helium abundance, and diatomic Hydrogen abundance.
  - Call `checkaddmm` from readatm.c to calculate mean molecular mass and check that the sum of abundances is within the permitted range of one. If not, raise a warning.

  - For each isotope, call `stateeqnford` **(FINDME: where is this defined?)** to calculate densities using the ideal gas law.
  - Increment to the next radius layer.
- Reallocate the arrays down to the final size according to the number of radius layers incremented in the infinite loop.
- Return the number of radius layers.

6.6.6 getmoldata:

Variables Modified
- Fill in `tr.ds.mol.radius, tr.ds.mol.ID, tr.ds.mol.mass` from the molecular information file.

Walkthrough
- Call `verbfileopen` from messagep.c to open the molecular info file if it exists.
- Skip past all comments and blank lines.
- Count the number of species
- Allocate arrays for molecule ID, mass, names, and radii.
- Skip past all comments and blank lines.
- Read molecular info from file by calling `getname` and `nextfield` from iomisc.c. Place into the allocated arrays.
- Loop over each molecule.
  - Call `findstring` from iomisc.c to check if the molecule's name matches any aliases from the file. If so, use the alias as the molecule's name. Otherwise, use the molecule's name from the molecule structure.
  - Call `findstring` from iomisc.c to find the index of the molecule. Use that index to set the radius, molecular ID, and mass in the molecule structure.

6.6.7 reloadatm:

Variables Modified
- Set `tr.ds.at.rads.i, tr.ds.at.rads.f` (initial and final radius sampling values) according to the new radius array.

Walkthrough
- Update temperature array at every layer.
- Update abundance array at every layer and for every molecule.
- Call `checkaddmm` to recalculate mean molecular mass and check whether the sum of abundances is sufficiently close to one. If not, print a warning.
- Check that radius reference level, pressure reference level, and surface gravity were defined. If not, raise an error.
- Call `radpress` from readatm.c to recalculate the radius array
- Set the initial radius value and final radius values according to the new radius array.
- Call `makeradsample` to make a new radius sampling array.

6.6.8 radpress:

Variables Modified

- Recalculate `tr.ds.at.rads.v`.

Walkthrough

- Set the first element of the radius array to 0.
- Loop over each radius layer.
  - Use cumulative trapezoidal integration to fill out the rest of the radius array using Equation 35.
  - Find the indices of the layers with pressures just above and below the reference pressure.
- Raise an error if the reference pressure was not found to be between any two layers in the pressure array and return 0.
- Log-linearly interpolate (linear in radius, logarithmic in pressure) to find the radius at the reference pressure.
- Shift the radius array to force the radius at the reference pressure equal to the reference radius.

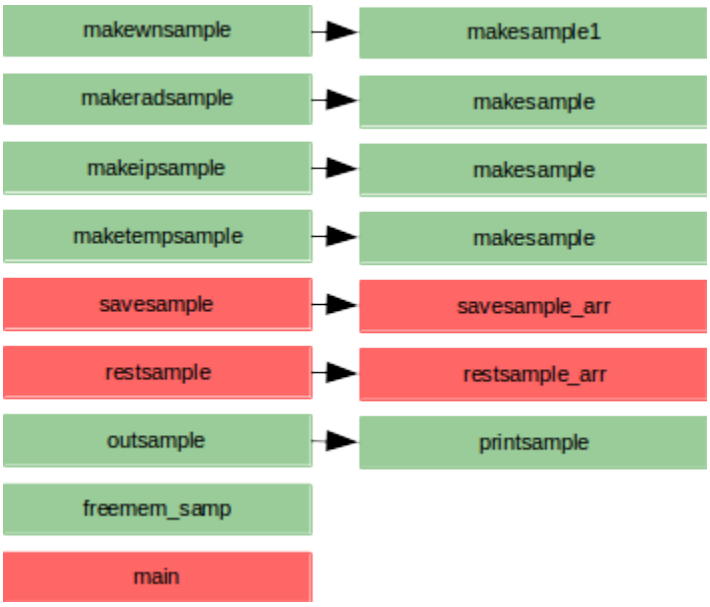

Figure 6: Function structure of makesample.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, green functions are used in both, and red functions are unused at this time.

6.7 makesample.c:

This file is concerned with producing sampling arrays for parameters including wavenumber, radius, temperature, and impact parameter. Sampling functions for each parameter call either `makesample` or `makesample1` to with the proper variables to create the sampling. Functions `savesample, savesample_arr, restsample, restsample_arr` are unused. `main` is only used for debugging. Function structure is shown in Figure 6.

6.7.1 List of Functions Defined in makesample.c:

`int makesample1(prop_samp *samp, prop_samp *ref, const long fl)`
Create a sampling array. Take values from a reference sampling.

`int makesample(prop_samp *samp, prop_samp *hint, prop_samp *ref, const long fl)`
Create a sampling array. Take values from hint or else from a reference sampling.

`int makewnsample(struct transit *tr)`
Call makesample to create the wavenumber sampling using the inverse-wavelength values as reference.

`int makeradsample(struct transit *tr)`
Call makesample to create the radius sampling.

`int makeipsample(struct transit *tr)`
Call makesample to create the impact parameter sampling using the reversed radius limits

and spacing as reference (always produce an equispaced sampling).

`int maketempsample(struct transit *tr)`
Call makesample to create the temperature sampling.

`static void printsample(FILE *out, prop_samp *samp, char *desc, long fl)`
Print a sampling's information to file.

`void savesample(FILE *out, prop_samp *samp)`
Save in binary the sample structure.

`void savesample_arr(FILE *out, prop_samp *samp)`
Saves in binary the sample structure's arrays.

`int restsample(FILE *in, prop_samp *samp)`
Restore a binary sample structure.

`int restsample_arr(FILE *in, prop_samp *samp)`
Restore a binary sample structure.

`int outsample(struct transit *tr)`
Print the sample data to file.

`void freemem_samp(prop_samp *samp)`
Frees the sampling structure.

`int main(int argc, char *argv[])`
De-bugging.

6.7.2 makesample1:

Variables Modified
- **(FINDME: this function allocates and modifies sampling stuff but does it for a number of parameters. Should we say those variables are modified here, or in the function which calls this one?)**

Walkthrough
- Set the acceptable ratio that the final value must fall in to not be truncated.
- Get sampling units factor, initial value, and final value from the given reference sampling.
- Raise an error and return -3 if the final value is less than the initial value.
- Raise an error and return -5 if the reference sampling has no spacing.
- If the reference sampling has spacing, set the sampling spacing equal to the reference spacing.
- If the spacing is negative, switch the sign on the acceptable ratio.
- Set the number of points for the sampling.
- Ensure that the number of points is positive.

- Check that the reference sampling has a valid oversampling factor (positive). If not, raise an error and return -6.
- Set the oversampling factor from the reference oversampling factor.
- Calculate the number of oversampled points and the spacing between oversampled points.
- Allocate and fill in sampling values.
- Check that the final sampling point coincides with the final value. If not, raise a warning.
- Return 0 (res is 0 for all cases) on success.

6.7.3 makesample:

Variables Modified
- **(FINDME: this function allocates and modifies sampling stuff but does it for a number of parameters. Should we say those variables are modified here, or in the function which calls this one?)**

Walkthrough
- Set the acceptable ratio that the final value must fall in to not be truncated.
- Get sampling units factor from the reference sampling if the hinted sampling is unset or invalid. Otherwise, use the hinted sampling.
- Get inital and final sampling values from hinted sampling. If hinted sampling is unset or invalid, get them from the reference sampling and update a flag to make note of this.
- Raise an error and return -5 if the reference sampling has no spacing.
- If the reference sampling has spacing:
  - Set the sampling spacing equal to the reference spacing.
- If the reference sampling does not have spacing:
  - If the initial and/or final values were taken from the reference sampling rather than the hinted sampling, raise a warning that this happened and that the initial or final values may have been modified.
  - Set the number of samples from the reference sampling.
  - Set the sampling spacing to 0.
  - Allocate sampling values, and copy from reference sampling values.
  - If an oversampling factor was given, raise a warning that this factor will be ignored.
  - Set oversampling factor to 0.
  - Return a flag indicating whether the reference inital and final values were used or not.
- If a spacing was hinted:
  - Set sampling spacing to hinted spacing.
- If none of these spacing conditions are true, raise an error that the sampling inputs are invalid.
- Raise an error and return -3 if the accepted inital and final sampling values create an invalid (zero or negative) interval.
- If the sampling spacing is negative, switch the sign on the acceptable ratio.
- Set the number of points for the sampling.
- Ensure that the number of points is positive.
- If the hinted oversampling factor is not given or invalid:
  - If the reference oversampling factor is not given or invalid, raise an error and return -6.
- If the hinted oversampling factor is valid, set the sampling oversampling factor equal to the hinted oversampling factor.

- Calculate the number of oversampled points and the oversampled spacing.
- Allocate and fill in sampling values.
- Check that the final sampling point coincides with the final value. If not, raise a warning.
- Return a flag indicating whether the reference initial and final values were used or not.

6.7.4 makewnsample:

Variables Modified:
- Call to makesample from makesample.c to set tr.wns values.
- Modify tr.pi to account for TRPI_MAKEWN.

Walkthrough
- If the hinted inital wavenumber sampling value is positive:
  - If the hinted wavenumber sampling factor is negative, raise an error.
  - Set the reference initial wavenumber sampling value from the hinted initial wavenumber sampling value.
- Otherwise, if the hinted initial wavelength sampling value is positive:
  - If the hinted wavenumber sampling factor is negative, raise an error.
  - Set the reference initial wavelength sampling value from the hinted initial wavelength sampling value.
- Otherwise, if no valid inital wavenumber or wavelength were given, raise an error.
- If the hinted final wavenumber sampling value is positive:
  - If the hinted wavenumber sampling factor is negative, raise an error.
  - Set the reference final wavenumber sampling value from the hinted final wavenumber sampling value.
- Otherwise, if the hinted final wavelength sampling value is positive:
  - If the hinted wavenumber sampling factor is negative, raise an error.
  - Set the reference final wavelength sampling value from the hinted final wavelength sampling value.
- Otherwise, if no valid final wavenumber or wavelength were given, raise an error.
- Set reference oversampling factor from hinted oversampling factor.
- Set reference unit conversion factor (1).
- Set reference number of samples to 0.
- Raise an error if no hinted sampling spacing is given.
- Set reference sampling spacing from hinted sampling spacing.
- Call makesample1 from makesample.c to make the oversampled wavenumber sampling.
- Set reference oversampling factor to 1 (no oversampling).
- Call makesample1 from makesample.c to make the wavenumber sampling.
- Call divisors from iomisc.c to calculate the exact divisors of the oversampling factor.
- Update progress indicator if sampling was successful.
- Return the result of makesample1.

6.7.5 makeradsample:

This function makes the radius sample. Take values from hint or else from the atmospheric file. Then the temperature, pressure, mean molecular mass, itostopes' density, abundance, partition function, and cross section are also resampled are resampled into an using a linear or spline interpolation, in case the radius array differ from the atmospheric radius array (i.e.,

hint given).

Variables Modified:
- Call to makesample to set tr.rads values.
- Allocate tr.ds.iso.isov.d, tr.ds.iso.isov.q, tr.ds.iso.isov.z (isotope's density, abundance, partition function).
- Set tr.ds.atm.tfct, tr.ds.atm.pfct from tr.ds.at.atm.tfct, tr.ds.at.atm.pfct (atmospheric unit factor for temperature and pressure).
- Allocate tr.atm.t, tr.atm.p, tr.atm.mm (transit's atmospheric temperature, pressure, and mean molecular mass).
- Set tr.atm.t, tr.atm.p, tr.atm.mm interpolating tr.ds.at.atm values into tr.rads sampling.
- Set tr.ds.iso.isov.d, tr.ds.iso.isov.q interpolating tr.ds.at.isov values into tr.rads sampling.
- Set tr.ds.iso.isov.c, tr.ds.iso.isov.z interpolating tr.ds.at.isov values into tr.atm.t array.
- Modify tr.pi to account for TRPI_MAKERAD.

Walkthrough:
- Set the reference sampling equal to the atmospheric structure sampling.
- Check that getatm and readinfo_tli have been executed.
- If a radius sample has already been generated, free the needed memory and unset the corresponding flag.
- Set flag to define linear or spline interpolation.
- If there is only one reference sampling (atmospheric sampling) point:
  - Set all radius sampling parameters to those in the atmospheric radius sampling structure. Allocate and set sampling values.
  - Set result flag to 0.
- Otherwise, if no hinted radius sampling spacing is given:
  - Set all radius sampling parameters to those in the atmospheric radius sampling structure. Allocate and set sampling values.
  - Set result flag to 0.
- Otherwise call makesample from makesample.c to make the radius sampling.
- Allocate arrays for molecular density and abundance, and set the number of layers for each molecule.
- Allocate array for partition function and set the number of layers for each isotope.
- Allocate arrays for atmospheric temperature, pressure, and mean molecular mass.
- Call resamplex from sampling.c to interpolate the radius sampling.
- Call resampley from sampling.c to interpolate the atmospheric pressure, temperature, and mean molecular mass.
- Call resample_free from sampling.c to free the resampling arrays.
- Loop over each database (species):
  - Call resamplex from sampling.c to interpolate temperatures from the TLI file.
  - Loop over each isotope:
    - Call resampley from sampling.c to interpolate the partition function from the TLI file.

- Call `resample_free` from sampling.c to free the resampling arrays.
- If sampling was successful, update the progress indicator.
- Return the result flag.

6.7.6 makeipsample:

This function makes the impact parameter sampling that determines the radii at which the planet probed for the transit geometry. Must be a decreasing array. If there is no hinted values, it uses the reversed radius array.

Variables Modified:
- Call to `makesample` from makesample.c to set `tr.ips` values.
- Modify `tr.pi` to account for `TRPI_MAKEIP`.

Walkthrough:
- If the hinted radius sampling spacing is -1:
  - Set impact parameter sampling from radius sampling, but reverse the values array.
- Otherwise:
  - Create impact parameter sampling from the hinted sampling parameters.
  - Create reference impact parameter sampling from the radius sampling.
  - Raise an error if the hinted final sampling value is less than the initial sampling value.
  - Check that `makeipsample, makeradsample` have been called.
  - Call `makesample` from makesample.c to create the impact parameter sampling.
- If desired, call `outsample` from makesample.c to print sample information to a file.
- Update the progress indicator if sampling was successful.
- Return the result flag.

6.7.7 maketempsample:

Variables Modified
- Call to `makesample` from makesample.c to set `tr.temps` values.
- Update `tr.pi` to account for `TRPI_MAKEIP`.

Walkthrough
- Create temperature sampling from hinted sampling parameters.
- Create an empty reference temperature sample.
- Raise an error if the final sampling value is less than the initial sampling value.
- Call `makesample` from makesample.c to create the temperature sampling.
- Update the progress indicator if sampling was successful.
- Return the result flag.

6.7.8 outsample:

Walkthrough
- Check that a filename exists. If not, return 0.
- If the filename is default and cannot be opened, raise a warning and return 1.
- Call `printsample` from makesample.c to print the following sampling structures: wavenumber, wavelength, radius, and impact parameter.

- Close the file.
- Return 0 on success.

6.7.9 printsample:

Walkthrough
- Print file header.
- Print sampling factor, inital value, final value, and spacing to file.
- Print oversampling to file if necessary.
- Print number of array elements to file.
- Print sampling values array to file.

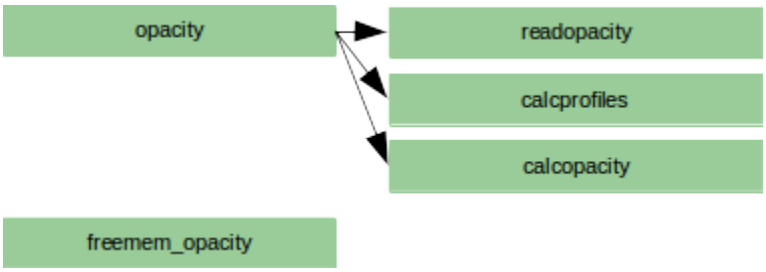

Figure 7: Function structure of opacity.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, green functions are used in both, and red functions are unused at this time.

6.8 opacity.c: EDITED

This file contains routines which calculate opacities, read opacity files, and write opacity files. Figure 7 shows the function structure.

6.8.1 List of Functions Defined in opacity.c:

`int opacity(struct transit *tr)`
Driver routine to calculate or read the opacity.

`int calcprofiles(struct transit *tr)`
Calculate a grid of Voigt profiles.

`int calcopacity(struct transit *tr, FILE *fp)`
Calculate a grid of opacities and Voigt profiles.

`int readopacity(struct transit *tr, FILE *fp)`
Read an opacity grid from file.

`int extinction(struct transit *tr, int r, int t)`
Calculate the opacity spectrum at a specific layer.

`int freemem_opacity(struct opacity *op, long *pi)`
Free index of refraction array.

6.8.2 opacity:

Modified
- Copy `th.f_opa` into `tr.f_opa`

Walkthrough
- Check that the radius array has been sampled.
- Check if an opacity file was specified.

- Call `fileexistopen` to check if an opacity file exists and if so, open it.
- Set the opacity file name in the transit structure from the hint structure.
- Call `readopacity` to read the opacity file if it exists.
- If the opacity file does not exist:
  - Open a file for writing.
  - Call `calcopacity` from opacity.c to calculate Voigt profiles and the opacity grid if requested.
- Update the progress indicator to account for `TRPI_OPACITY`.
- Return 0 on success.

6.8.3 calcprofiles:

Modified
- Copy `tr.ds.th.nDop, tr.ds.th.nLor` into `tr.ds.op.nDop, tr.ds.op.nLor`.
- Set `tr.ds.op.aDop, tr.ds.op.aLor` equal to logspaces from given minimum and maximum (`tr.ds.th.dmin, tr.ds.th.dmax, tr.ds.th.lmin, tr.ds.th.lmax`).
- Allocate `tr.ds.op.profsize` (Voigt profile half-size).
- Allocate `tr.ds.op.profile` (Voigt profiles).
- Call `getprofile` from extinction.c to fill out `tr.ds.op.profsize`.

Walkthrough
- Make a logscale grid for the profile widths according to given min and max values.
- Allocate an array for the profile half-size.
- Allocate grid of Voigt profiles.
- Loop over all Doppler and Lorentz widths to calculate Voigt profiles
  - If the Doppler width is an order of magnitude smaller than the Lorentz width, and this is not the first calculation performed, set the profile half-size equal to the previous profile(skipping the calculation)
  - Otherwise, call to `getprofile` in extinction.c to calculate Voigt profile half-size.
- Return 0 on success.

6.8.4 calcopacity:

Modified
- Copy `tr.temp.n` into `tr.ds.op.Ntemp`.
- Allocate `tr.ds.op.temp` (temperature array) and copy from `tr.temp.v`.
- Allocate and evaluate `tr.ds.op.ziso` (Partition function for each isotope and temperature).
- Copy `tr.rads.n` into `tr.ds.op.Nlayer` (number of radius layers).
- Allocate `tr.ds.op.press` (pressure array) and copy from `tr.atm.p` in CGS units.
- Copy `tr.ds.iso.nmol` into `tr.ds.op.Nmol` (number of molecules).
- Allocate `tr.ds.op.molID` (molecule IDs).
- Add molecule IDs to `tr.ds.op.molID` if not there.
- Copy `tr.wns.n` into `tr.ds.op.Nwave` (number of wavenumber samples).
- Allocate `tr.ds.op.wns` and copy from `tr.wns.v` (wavenumber samples).
- Allocate `tr.ds.op.o` (4D opacity array).

Walkthrough
- Call `maketempsample` from makesample.c to create a temperature array from hinted values and put the temperature array in the opacity structure.
- Allocate the partition function array.
- Set the interpolation function flag.
- Interpolate the isotope partition function. (FINDME: be more specific here.)
- Get pressure array from the transit structure and place in the opacity structure.
- Get molecule array from the transit structure and place in the opacity structure.
- For each molecule, check if its ID is in the molecule ID array. If not, add it.
- Get wavenumber array from the transit structure and place in the opacity structure.
- Allocate the 4-dimensional opacity array ([mol][temp][rad][wn])
- For each radius layer and temperature, call to `extinction` in opacity.c to compute extinction.
- Write dimension sizes to file.
- Write molecular ID, temperature, pressure, and wavenumber sampling arrays to file.
- Write the opacity array to file.
- Close the file.
- Return 0 on success.

6.8.5 readopacity:

Modified
- Allocate `tr.ds.op.molID, tr.ds.op.temp, tr.ds.op.press, tr.ds.op.wns` and fill in from file.
- Allocate `tr.ds.op.o` and fill in from file.

Walkthrough
- Read the dimension sizes (number of molecules, temperatures, radius layers, and wavenumbers) from file.
- Allocate molecular ID, temperature, pressure, and wavenumber sampling arrays.
- Read molecular ID, temperature, pressure, and wavenumber sampling arrays from file.
- Allocate the 4D opacity grid.
- Read the opacity grid from file.
- Return 0 on success.

6.8.6 extinction:

Calculates the extinction coeffecient for a given radius layer and temperature.

Modified
- Calculate extinction and fill `tr.ds.op.o` for a single radius layer and temperature.

Walkthrough
- Calculate constant factors for Doppler and Lorentz line widths.
- Allocate temporary arrays for line widths, extinction, and line width indices.
- Loop over isotopes.
  - Loop over molecules.

    - Calculate the isotope's collision diameter. (FINDME: cross section?)
    - Call to `stateeqnford` to calculate the density of molecule colliding with the isotope. (FINDME: where is this defined?)
    - Calculate Lorentz line width for this molecule and add it to the total line width for this isotope.
  - Multiply the Lorentz line width for this isotope by the constant factor.
  - Calculate Doppler width divided by central wavenumber for this isotope.
  - Find the maximum between the Doppler and Lorentz widths.
  - Find the minimum between the maximum width and the previous minimum width. The first minimum width is set at the beginning of the function.
  - Call `binsearchapprox` from iomisc.c to perform a binary search to find the indices of the Doppler and Lorentz widths in the Doppler and Lorentz width samples.
- Set oversampling resolution by looping through the exact divisors of the oversampling factor until the divisor times the spacing of the finest oversampling is greater than half the width of the smallest profile.
- Loop over every line to calculate the maximum extinction coefficient for each molecule.
  - Calculate the wavenumber of the line transition.
  - Call `valueinarray` to find the molecule index in the opacity grid for this isotope.
  - Skip calculation for this line transition if it is not within the given limits.
  - Calculate the extinction coefficient divided by the molecular abundance.
  - If the maximum extinction for this molecule has not been calculated yet, set it equal to the extinction coefficient that was just calculated. Otherwise, set the maximum and minimum extinction for this molecule equal to the maximum and minimum between the recently calculated extinction and the previously calculated maximum and minimum.
- Loop over each line to calculate extinction coefficients.
  - Calculate the wavenumber of the line transition.
  - Call `valueinarray` to find the molecule index in the opacity grid for this isotope.
  - Skip calculation for this line transition if it is not within the given limits.
  - Calculate the extinction coefficient divided by the molecular abundance (FINDME: reference equation).
  - Find the index of the closest oversampled wavenumber.
  - Check if the next line falls within the same sampling unit (same sampling index). If so, co-add the next line with the current line (add the next line's extinction to the opacity for this line) and skip the next line's calculations.
  - If the extinction for this line is less than the defined threshold factor times the maximum extinction, disregard this line and continue to the next.
  - Calculate the closest dynamic sampling wavenumber.
  - Check if the ratio of Doppler width to Lorentz width is greater than a given threshold. If so, call to `binsearchapprox` to do a binary search to recalculate the index for the Doppler width. If not, then the exact width of the Doppler profile is unimportant and the calculation is skipped.
  - Calculate the offset between the center of the line and the dynamic wavenumber sample (in units of oversampled wavenumber spacing).
  - Calculate the offset between the edge of the profile and the beginning of the wavenumber array (in units of oversampled wavenumber spacing)
  - Calculate the lower and upper indices of the profile (in units of dynamically sampled wavenumber)

    - Fix the lower and upper indices to the boundaries if they go outside the bounds of the wavenumber sampling.
    - Add the contribution from this line (and any co-added lines) to the opacity spectrum.
- Call `downsample` to downsample the temporary extinction array to the final sampling size and fill in the opacity grid.
- Free all temporary arrays.
- Return 0 on success.

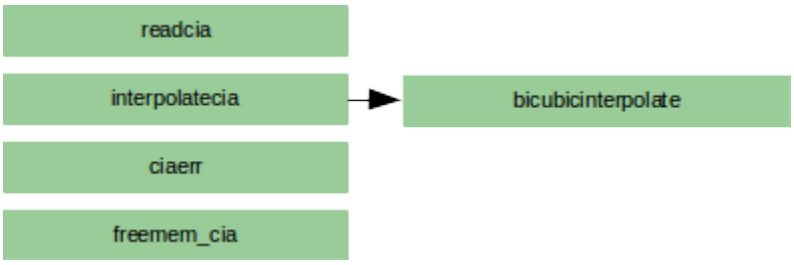

Figure 8: Function structure of cia.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, and green functions are used in both.

6.9 cia.c: EDITED

This file contains routines which are used to read the CIA files(s) and interpolate the values therein to the `transit` sampling. Figure 8 shows the function structure of cia.c.

6.9.1 List of Functions Defined in cia.c:

```
int readcia(struct transit *tr)
```
Read CIA info from tabulated files.

```
int interpolatecia(struct transit *tr)
```
Get number of CIA files from hint. Allocate tr.ds.cia variables. Open files, read isotope names and sampled temperatures. Read tabulated data (wavenumber x temperatures). Interpolate values from tabulated sample to transit sample. Get density arrays of the isotopes from transit. Calculate absorption coefficients in cm^{-1}.

```
int bicubicinterpolate(double **res, double **src, double *x1, long nx1,
                       double *x2, long nx2, double *t1, long nt1,
                       double *t2, long nt2)
```
Interpolates 'src' into 'res' according to the new dimensions, first interpolates the second dimension and then the first. The result is added to 'res'.

```
void ciaerr(int max, char *name, int line)
```
Error printing function for lines longer than maxline in the CIA file.

```
int freemem_cia(struct cia *cia, long *pi)
```
Free cia structure.

6.9.2 readcia:

Modified
- Copy `tr.ds.th.ncia` into `tr.ds.cia.ncia`.
- Fill in `tr.ds.cia.cia` from file.
- Update `tr.pi` to account for `TRPI_CIA`.

Walkthrough

- Check that radius and wavenumber samples have been made.
- Allocate extinction array in static cia structure.
- If there are no CIA files, return 0.
- Allocate molecule names, molecule IDs, number of temperatures and wavenumber samples per file, and CIA array.
- Loop over each CIA file (each molecule pair):
  - Read the file name from the transit hint structure.
  - Open the file.
  - Skip any comments and blank lines at the top of the file.
  - When an 'i' character is encountered:
    - If pointing to a blank space, increment the pointer to the next character.
    - Count the number of words in the line. If not 2, raise an error.
    - Copy the name of the first molecule.
    - Find the ID of the first molecule by comparing its name with the molecule IDs.
    - Raise an error if the molecule from file does not match any IDs.
    - Copy the name of the second molecule.
    - Find the ID of the second molecule by comparing its name with the molecule IDs.
    - Raise an error if the molecule from file does not match any IDs.
    - Continue reading the file.
  - When a 't' character is encountered:
    - If pointing to a blank space, increment the pointer to the next character.
    - Count the number of temperature samples in the file.
    - Raise an error if no temperature samples are found.
    - Loop over the temperatures and copy them into the static cia temperature array.
    - Continue reading the file.
  - Set the initial value for allocated wavenumber fields. This must be a power of 2 for sizing purposes.
  - Allocate the wavenumber and extinction arrays.
  - Begin infinite loop to read in data:
    - Increment the pointer past all comments and blank lines.
    - Check if the end of the file has been reached. If so, break the loop.
    - Check if the number of read wavelengths is equal to the allocated wavenumber fields. If so, reallocate the array to double its size by bitwise left-shifting the number of fields. Since this was initally set to a power of two, a bitwise left-shift will double the value.
    - Increment the pointer past all blank spaces
    - Read in the wavenumber at pointer location.
    - Loop over each temperature and copy the corresponding extinction value to the extinction array.
    - Increment looping indices.
  - Reallocate the arrays to remove extra rows added when doubling the size.
  - Store the extinction array in the CIA structure.
  - Close the file.
- Update the progress indicator to account for `TRPI_CIA`.
- Return 0 on success.

6.9.3 interpolatecia:

Modified
- Fill out `tr.ds.cia.e` by calling `bicubicinterpolate` for each CIA file.

Walkthrough
- Allocate temporary temperature and wavenumber arrays.
- Reset CIA opacity to zero.
- Allocate temporary array for opacity.
- Set tempoarary temperature and wavenumber arrays from the transit structure.
- For each CIA file:
  - Call `bicubicinterpolate` from cia.c to interpolate CIA data to the wavenumber and temperature sampling.
  - Get density profiles of isotopes from molecular information structure.
  - Calculate CIA absorption coefficients at each radius and wavenumber.
- Free temporary arrays.
- Return 0 on success.

6.9.4 bicubicinterpolate:

Walkthrough
- Set the first and last values of the source array.
- Check that the sampling regions match. If not, return 0.
- Find indices where the target array is within the source array boundaries (so that the result is an interpolation, not an extrapolation).
- Use GSL to perform cubic interpolation over the first index.
- Use GSL to perform cubic interpolation over the second index.
- Free temporary arrays.
- Return 0.

6.10 idxrefraction.c:

This file is concerned with calculating the index of refraction at each radius level. Currently this is 1 at all layers (no light bending). Functions `restidxref, saveidxref` are unused.

6.10.1 List of Functions Defined in idxrefraction.c:

`int idxrefrac(struct transit *tr)`
Calculates the index of refraction. Currently, it sets an index of refraction of 1.0 at all levels (no light bending).

`int freemem_idexrefrac(struct idxref *ir, long *pi)`
Free index of refraction array.

`int restidxref(FILE *in, PREC_NREC nrad, struct idxref *ir)`
Restore hints structure, the structure needs to have been allocated before.

`void saveidxref(FILE *out, PREC_NREC nrad, struct idxref *ir)`
Write index of refraction values to file pointed by out.

6.10.2 idxrefrac:

Variables Modified
- Allocate and set values of `tr.ds.ir.n` (Index of refraction per radius array)
- Update `tr.pi` to account for `TRPI_IDXREFRAC`.

Walkthrough
- Call to `transitcheckcalled` in transitstd.c to check that `makeradsample` has been called.
- Allocate index of refraction array.
- Loop over each radius layer:
  - Call to `stateeqnford` **(FINDME: from where?)** to calculate density.
  - Calculate index of refraction (currently always 1).
- Return 0 on success.

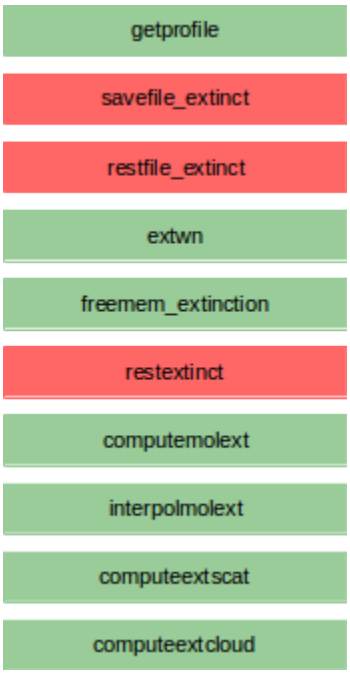

Figure 9: Function structure of opacity.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, green functions are used in both, and red functions are unused at this time.

6.11 extinction.c: EDITED

This file contains routines associated with computing molecular extinction. That includes a wrapper function to calculate Voigt profiles, a function to compute molecular extinction, and a function to interpolate the molecular extinction. There are also functions which compute extinction from other sources (`computeextscat, computeextcloud`) although they are not fully implemented. Functions `savefile_extinct, restfile_extinct, restextinct` are unused. Figure 9 shows the function structure of extinction.c.

6.11.1 List of Functions Defined in extinction.c:

```
inline int getprofile(PREC_VOIGT **pr, int vf, PREC_RES dwn, PREC_VOIGT dop,
                      PREC_VOIGT lor, float ta)
```

Driver to calculate a Voigt profile.

```
void savefile_extinct(char *filename, PREC_RES **e, _Bool *c, long nrad,
                      long nwav)
```

Saving extinction for a possible next run

```
void restfile_extinct(char *filename, PREC_RES **e, _Bool *c, long nrad,
                      long nwav)
```

Restoring extinction for a possible next run

```
int extwn(struct transit *tr)
```

Fill up the extinction information

`void printone(struct transit *tr)`
Printout for one P,T conditions

`int freemem_extinction(struct extinction *ex, long *pi)`
Free extinction coefficient structure arrays

```
int restextinct(FILE *in, PREC_NREC nrad, short niso, PREC_NREC nwn,
                struct extinction *ex)
```
Restore hints structure, the structure needs to have been allocated before

`int computemolext(struct transit *tr, PREC_NREC r, PREC_RES **kiso)`
Compute the molecular extinction.

`int interpolmolext(struct transit *tr, PREC_NREC r, PREC_RES **kiso)`
Interpolate the opacity grid at the specified layer to obtain molecular extinction.

`void computeextscat(double *e, long n, struct extscat *sc, double *rad, double trad, double *temp, double tcft, double wn)`
Compute scattering contribution to extinction. Currently fills the scattering extinction array with 0's (no extinction due to scattering).

`void computeextcloud(double *e, long n, struct extscat *sc, double *rad, double trad, double *temp, double tcft, double wn)`
Compute cloud contribution to extinction. Cloud extinction is linear, increasing from top of the clouds to the bottom of the clouds.

6.11.2 getprofile

Variables Modified
- Allocate `op.profile` (**pr).

Walkthrough
- Find the largest width between Doppler and Lorentz
- Calculate the range for computation in half-widths.
- Calculate the number of points in the profile.
- Check that the profile contains at least 3 elements. If not, set to 3.
- If the profile is larger than the wavenumber range, shrink the profile.
- Allocate the profile array.
- Calculate the Voigt profile using a width that gives and integer number of `dwn` spaced bins.
- Return the number of points in half the profile.

6.11.3 extwn:

Modified:
- Copy `th.ethresh` into `ex.ethresh` (extinction threshold).

- Allocate `ex.e` (extinction).
- Allocate `ex.computed` (boolean to indicate if extinction has been computed at the corresponding radius layer).
- Update `tr.pi` to account for `TRPI_EXTWN`.

Walkthrough:
- Check that `readlineinfo_tli, readdatarng, makewnsample, makeradsample` have been executed.
- **(FINDME: There's a note here that some of this function should be in readatm)**
- Set extinction coefficient threshold from transithint structure.
- Allocate extinction coefficient array.
- Allocate boolean for checing if extinction has been computed.
- Update progress indicator to account for `TRPI_EXTWN`.

6.11.4 computemolext:

This routine computes the molecular extinction coefficient (e_m, in cm^{-1}) at one specific atmospheric radius, Equations (3.36)–(3.37) of P. Rojo's thesis (see also Equation 1). Initially, the code calculates the Doppler and Lorentz line-broadening widths (Equations 2 and 3), to later calculate the Voigt profile.

Modified
- Calculate `tr.ds.ex.e` (Extinction coefficient) for the given radius layer.
- Set `tr.ds.ex.computed` of given radius to True.

Walkthrough
- Allocate a temporary extinction array.
- Calculate the dynamic wavenumber sampling interval and the oversampled dynamic wavenumber sampling interval.
- Calculate constant factors for Doppler and Lorentz line widths.
- Allocate arrays for the Doppler and Lorentz line widths and arrays for line width indices.
- Loop over each isotope.
  - Loop over each molecular species.
    - Calculate the isotope's collisional cross-section with this molecule and add the resulting Lorentz width to the Lorentz width for this isotope.
  - Multiply by the constant factor to get the Lorentz width for this isotope.
  - Calculate the Doppler width divided by the central wavenumber (because Doppler width is wavenumber-dependent).
  - Find the maximum between the Lorentz width and Doppler width.
  - Find the minimum between this maximum and the previously calculated minimum (this minimum is set to the maximum between the widths on the first iteration).
  - Call `binsearchapprox` from iomisc.c to perform a binary search to find the indices of the Doppler and Lorentz widths in the Doppler and Lorentz width samples.
- Set oversampling resolution by looping through the exact divisors of the oversampling factor until the divisor times the spacing of the finest oversampling is greater than half the width of the smallest profile.
- Loop over every line to calculate the maximum extinction coefficient for each molecule.

  - Calculate the wavenumber of the line transition.
  - Skip calculation for this line transition if it is not within the given limits.
  - Calculate the extinction coefficient except the broadening factor. (FINDME: ?)
  - If the maximum extinction for this molecule has not been calculated yet, set it equal to the extinction coefficient that was just calculated. Otherwise, set the maximum and minimum extinction for this molecule equal to the maximum and minimum between the recently calculated extinction and the previously calculated maximum and minimum.
- Loop over each line to calculate extinction coefficients.
  - Calculate the wavenumber of the line transition.
  - Skip calculation for this line transition if it is not within the given limits.
  - Calculate the extinction coefficient. (FINDME: reference equation)
  - Find the index of the closest oversampled wavenumber.
  - Check if the next line falls within the same sampling unit (same sampling index). If so, co-add the next line with the current line (add the next line's extinction to the opacity for this line) and skip the next line's calculations.
  - If the extinction for this line is less than the defined threshold factor times the maximum extinction, disregard this line and continue to the next.
  - Calculate the closest dynamic sampling wavenumber.
  - Check if the ratio of Doppler width to Lorentz width is greater than a given threshold. If so, call to `binsearchapprox` to do a binary search to recalculate the index for the Doppler width. If not, then the exact width of the Doppler profile is unimportant and the calculation is skipped.
  - Calculate the offset between the center of the line and the dynamic wavenumber sample (in units of oversampled wavenumber spacing).
  - Calculate the offset between the edge of the profile and the beginning of the wavenumber array (in units of oversampled wavenumber spacing).
  - Calculate the lower and upper indices of the profile (in units of dynamically sampled wavenumber)
  - Fix the lower and upper indices to the boundaries if they go outside the bounds of the wavenumber sampling.
  - Add the contribution from this line (and any co-added lines) to the opacity spectrum.
- Call `downsample` to downsample the temporary extinction array to the final sampling size and fill in the extinction array for this radius.
- Free all temporary arrays.
- Update the boolean that indicates extinction has been computer for this layer.
- Return 0 on success.

6.11.5 interpolmolext:

Modified
- Fill in `tr.ds.ex.e`.
- Set the radius index of `tr.ds.ex.computed` equal to 1.

Walkthrough
- Perform a binary search to find the index of grid-temperature immediately lower than layer temperature.
- Loop over wavenumber

    - Loop over molecules
        - Calculate extinction coefficient by linear interpolation of the opacity grid between the index found by the binary search and the next one.
        - Call `valueinarray` to find the index of the molecule.
        - Add the extinction for this molecule to extinction
- Update boolean to show extinction has been computed.
- Return 0 on success.

6.11.6 computeextscat:

Modified
- Fill out scattering extinction array (passed to function, not found in any structures).

Walkthrough
- Loop over each radius layer. Set scattering extinction to 0 at all layers.

6.11.7 computeextcloud:

Modified
- Fill out cloud extinction array (passed to function, not found in any structures).

Walkthrough
- If there are no clouds, set the cloud extinction array to zero everywhere.
- Calculate the amount of extinction per distance due to the clouds.
- Loop down through the radius layers, setting the cloud extinction to 0 until reaching the clouds.
- Loop down through the radius layers, starting from the top of the clouds, setting the cloud extinction to a linearly increasing amount according to the extinction per distance until reaching the bottom of the clouds.
- Loop through the remaining radius layers, setting the cloud extinction to the total extinction due to clouds.

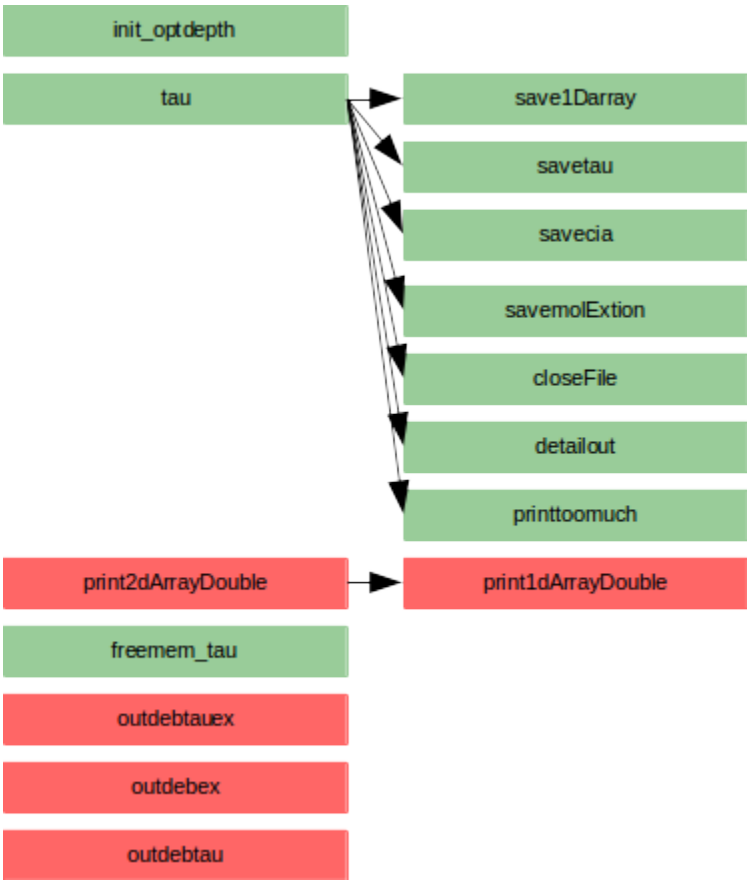

Figure 10: Function structure of tau.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, green functions are used in both, and red functions are unused at this time.

6.12 tau.c: EDITED

This file contains all routines associated with calculation of optical depth. Functions `detailout`, `outdebtauex`, `outdebex`, and `outdebtau` are unused. Functions `print2dArrayDouble`, `print1dArrayDouble` are generic print-to-file functions that are not generally used in `transit`. Figure 10 shows the function structure of tau.c. **(FINDME: reference the equation in BART theory doc)**

6.12.1 List of Functions Defined in tau.c:

`int init_optdepth(struct transit *tr)`
Initialize the optical depth structure for eclipse and transit geometry.

`int tau(struct transit *tr)`
Calculate the extinction coefficient and optical depth as a function of layer/impact parameter and wavelength.

`int detailout(prop_samp *wn, prop_samp *rad, struct detailfld *det, PREC_RES **arr, short flag)`
Print to file the optical depth, cia, or extinction at the requested wavenumbers (given by det).

`void printtoomuch(char *file, struct optdepth *tau, prop_samp *wn, prop_samp *rad)`
Print (to file or stdout) the impact parameter where the optical depth reached toomuch (for each wavenumber).

`int freemem_tau(struct optdepth *tau, long *pi)`
Free tau structure.

`void outdebtauex(char *name, PREC_RES **e, prop_samp *ip, PREC_RES **t, long rn, long w)`
Print to file (name) the optical depth and extinction as a function of impact parameter (up to layer index rn), for given wavenumber (with index w).

`void outdebex(char *name, PREC_RES **e, PREC_RES *r, long rn, long wi, long wf)`
Print to file (name) the extinction coefficient as a function of radius (up to layer index rn) for the specified wavenumber range (indices from wi to wf).

`void outdebtau(char *name, prop_samp *ip, PREC_RES **t, long wi, long wf)`
Print to file (name) the optical depth as function of impact parameter for the specified wavenumber range (indices from wi to wf).

6.12.2 tau:

Main routine where the extinction coefficient and the optical depth are calculated. This function sets up the optical depth parameters and then calls to the `computeextradius` and `totaltau` subroutines to do the calculations.

In the code, `transittau` or `eclipsetau` is pointed by the variable `fcn`. `transittau` is defined in the `transit_ray_solution slantpath` variable at the end of `slantpath.c`. `slantpath` is assigned to the transit variable `tr.sol` in the function `acceptgenhints` from `argum.c`. **(FINDME: update to include eclipse ray solution)**.

Variables Modified:
- Copy `tr.save.ext` from `th.save.ext` (extinction output filename).
- Call to `init_optdepth` to initialize `tr.tau`.
- Set `tr.cl.cloudext, tr.cl.cloudtop, tr.cl.cloudbottom` (Cloud maximum opacity, top layer radius, layer radius of cloudext).
- Call to `computemolext` or `interpolmolext` to calculate `tr.ds.ex.e` (extinction coefficient).
- Call to `eclipsetau` or `transittau` to calculate `tr.tau.t` (optical depth).
- Set `tr.tau.last` if `tr.tau.t > toomuch` (radius index of last calculated tau).
- Call to `savefile_extinct` to store `tr.ds.ex.e` in file.
- Call to `printtoomuch` to store the radius where the optical depth reached toomuch.

- Call to `freemem_lineinfotrans` to free `tr.ds.li` (line info struct).
- Call to `freemem_localextinction` to free `tr.ds.ex.e` and related static variables (extinction coefficient).
- Update `tr.pi` to account for `TRPI_TAU`.

Walkthrough:
- Store the height of each layer (eclipse) or impact parameter (transit) starting from the outermost layer in local variable h.
- Check that there are enough radius layers for interpolation (4+). If not, raise an error.
- Check that idxrefrac and extwn functions have been called.
- Pass TAU flags from transithint structure to transit structure.
- Declare arrays for cloud and scattering extinction (per wavenumber).
- Restore extinction save file (if requested).
- Compute molecular extinction at the outermost layer.
- Start loop, over wavenumber, to calculate the extinction:
  - Compute the scattering and cloud extinction for all layers at given wavenumber.
  - Start loop, over the layers/impact parameters:
    - Check if the molecular extinction has been calculated at this layer, if not calculate it for all wavenumbers at this layer.
    - Call transittau or eclipsetau (as fcn) to calculate the optical depth at given wavenumber and layer/impact parameter. See Equation 29.
    - If the optical depth reached toomuch, end the layer/impact-parameter loop.
- Print to file detailed output of tau, extinction, and CIA if requested.
- Print to file the lowest layer/impact parameter reached before optical depth reached toomuch.
- Update the progress indicator.
- Return 0 on success.

6.12.3 init_optdepth:

Variables Modified:
- Initialize `tr.ds.tau` and `tr.ds.intens`.
- Set `tr.tau.toomuch` from `th.toomuch` (max optical depth to calculate).
- Allocate `tr.tau.t, tr.tau.last` (optical depth and index of toomuch).
- Allocate `tr.ds.intens.a` (intensity grid).

Walkthrough:
- Allocate the optical depth structure.
- Pull maximum optical depth from transithint structure into optical depth structure.
- Allocate array for the layer index where tau reaches toomuch (max optical depth).
- Allocate the optical depth array
- Allocate the intensity grid structure and intensity array if using eclipse geometry.
- Return 0 on success.

6.12.4 detailout:

Walkthrough:

- Check that there a file name has been given.
- Perform a binary search to find the indices of the requested wavenumbers.
- Print wavenumber.
- Print radii and corresponding value.
- Close the file.
- Return 0 on success.

6.12.5 freemem_tau:

Variables Modified:
- Free tau.t and tau.last.
- Update `tr.pi` to remove for `TRPI_TAU`.

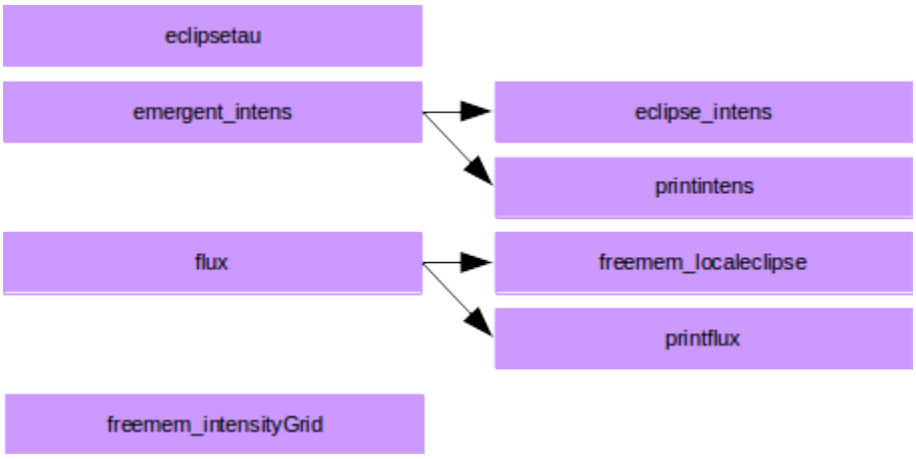

Figure 11: Function structure of eclipse.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, and green functions are used in both.

6.13 eclipse.c: EDITED

This file contains routines associated with calculating flux from an eclipse. This includes calculating optical depth at each wavenumber and incident angle, emergent intensity at each wavenumber, emergent intensity over all wavenumbers, and flux over all angles. Figure 11 shows the function structure of eclipse.c.

6.13.1 List of Functions Defined in eclipse.c:

`static PREC_RES eclipsetau(struct transit *tr, PREC_RES height, PREC_RES *ex)`
Computes optical depth for eclipse geometry for one ray and one wavenumber at various incident angles on the planet surface, between a certain layer in the atmosphere up to the top layer.

`static PREC_RES eclipse_intens(struct transit *tr, PREC_RES *tau, PREC_RES w, long last, double toomuch, prop_samp *rad)`
Calculates emergent intensity for one wavenumber.

`int emergent_intens(struct transit *tr)`
Driver function that calculates emergent intensity for the whole range of wavenumbers at various points on the planet.

`int flux(struct transit *tr)`
Calculates flux by integrating intensity over predefined angles.

`void printintens(struct transit *tr)`
Print (to file or stdout) the emergent intensities as a function of wavelength for each angle.

`void printflux(struct transit *tr)`
Print (to file or stdout) the flux as a function of wavenumber.

`freemem_localeclipse()`
Free eclipse pointer arrays.

`freemem_intensityGrid(struct grid *intens, long *pi)`
Free intensity grid structure arrays.

6.13.2 eclipsetau

Walkthrough
- Use a binary search to find the index of the sampled radius immediately below or equal to the height.
- Check if the sampled radius is the outer layer, and if so return 0.
- Move pointers to the location of height.
- Check that there are sufficient points for spline integration. If not, create them halfway between the given points.
- Calculate the distance along the path for each radius.
- Use trapezoidal integration along the path to calculate optical depth per unit radius.
- Return optical depth per unit radius.

6.13.3 eclipse_intens:

Walkthrough
- Calculate the Planck blackbody function for each radial layer. See Equation 32.
- Calculate the transmission function for each layer of the planet. This is the integrand of the integral in Equation 33.
- After tau reaches toomuch, fill remaining layers with 0 flux.
- Use GSL to integrate tau up to maximum tau. See Equation 33.
- Return integration result (intensity).

6.13.4 emergent_intens:

Variables Modified
- Call `eclipse_intens` to calculate `tr.ds.intens.a` (intensity[angle][wn]).
- Update `tr.pi` to account for `TRPI_MODULATION`.

Walkthrough
- Call `eclipse_intens` from eclipse.c as sol.spectrum to calculate the intensity at every wavenumber.
- Update the progress indicator to account for `TRPI_MODULATION`
- Call `printintens` from eclipse.c to print the emergent intensity as a function of wavenumber to file.
- Return 0 on success.

6.13.5 flux:

Variables Modified

- Allocate `tr.ds.out.o` (emergent flux) (FINDME: how to say this? It creates and fills out a local out structure).
- Calculate `tr.ds.out.o` (flux). See Equation 34.

Walkthrough

- Allocate area grid and fill (local variable).
- Allocate array for emergent flux.
- Calculate flux from intensity grid and area.
- Call freemem_localeclipse to free area grid.
- Call printflux to print the flux.
- Return 0 on success.

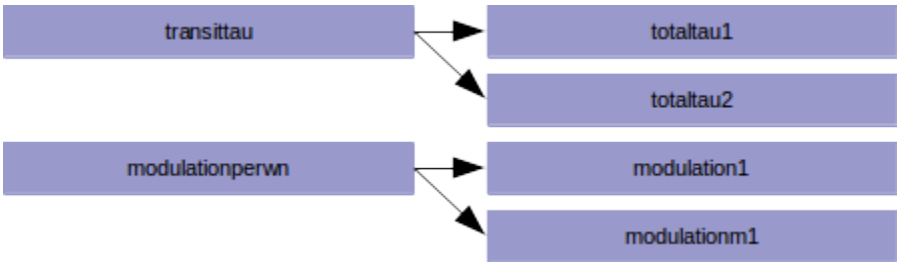

Figure 12: Function structure of slantpath.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, and green functions are used in both.

6.14 slantpath.c: PC-EDITED

This file contains routines that calculate tau at a specific impact parameter and wavenumber, and routines that calculate modulation for a specific wavenumber. totaltau2, which is intended to calculate tau taking into account a variable index of refraction, is unused and untested. Figure **??** shows the function structure of slantpath.c.

6.14.1 List of Functions Defined in slantpath.c:

```
static PREC_RES totaltau1(PREC_RES b, PREC_RES *rad, PREC_RES refr,
                          PREC_RES *ex, long nrad)
```

Compute the light path and optical depth at a given impact parameter and wavenumber, for a medium with constant index of refraction.

```
static PREC_RES totaltau2(PREC_RES b, PREC_RES *rad, PREC_RES *refr,
                          PREC_RES *ex, long nrad)
```

Compute the light path and optical depth at a given impact parameter and wavenumber, for a medium with variable index of refraction.

```
static inline PREC_RES transittau(PREC_RES b, PREC_RES *rad, PREC_RES *refr,
                                  PREC_RES *ex, long nrad, int exprlevel)
```

Driver function to calculate the optical depth at a given impact parameter at a specific wavenumber.

```
static PREC_RES modulationperwn(PREC_RES *tau, long last, double toomuch,
                                prop_samp *ip, struct geometry *sg,
                                int exprlevel)
```

Driver function to calculate the modulation in/out-of-transit ratio for a single wavenumber.

```
static PREC_RES modulation1(PREC_RES *tau, long last, double toomuch,
                            prop_samp *ip, struct geometry *sg)
```

Calculate the transit's modulation at a given wavenumber for no-limb darkening nor emitted flux.

```
static inline PREC_RES modulationm1(PREC_RES *tau, long last, double toomuch,
                                    prop_samp *ip, struct geometry *sg)
```

Calculate the modulation at a given wavenumber, considering the planet as an opaque disc

of radius r = r(tau=toomuch), for no-limb darkening nor planet emission.

6.14.2 totaltau1:

Walkthrough:
- Calculate the minimum distance of the ray path to the center of the planet (`r0`).
- Get the index (`rs`) of the sampled radius below or equal to `r0`.
- Move the extinction and radius pointers to `rs`.
- Calculate the extinction coefficient at the closest approach radius by parabolic interpolation.
- If there are only two elements in the extinction and radius arrays, create a 3rd temporary element between the two values.
- Calculate the distance along the lightray path.
- Calculate the optical depth by integrating (with GSL) the extinction along the ray path (up to the closest approach). See Equation 29.
- Reset the original values of the extinction and radius arrays (in case of 2 elements).
- Return the result of integration to account for full multiplied by 2.

6.14.3 totaltau2:

Walkthrough:
- Warn user that this routine is untested (and surely will not work).
- Calculate the minimum distance of the ray path to the center of the planet (`r0`).
- Get the index (`rs`) of the sampled radius below or equal to `r0`.
- Move the radius pointer to the element corresponding to the sampled radius index.
- Calculate the analytical part of the extinction integral.
- Calculate the optical depth by integrating (with GSL) if there are at least 3 points available. See Equation 29. **(FINDME: is this right?)**
- If there are only two points available, use Trapezium integration.
- Return the result of integration to account for full multiplied by 2.

6.14.4 transittau:

Variables Modified:
- Set `tr.taulevel` from `th.taulevel` (Constant or variable index of refraction per layer).

Walkthrough:
- Read the taulevel flag to determine a constant or variable index of refraction.
- Call to `totaltau1` or `totaltau2` depending on taulevel.
- Return the value given by `totaltau1` or `totaltau2`.

6.14.5 modulation1:

Walkthrough:
- Get the stellar radius.
- Calculate integrand of modulation. See Equation 31.
- Add a layer with an integrand value of 0.

- Raise an error if there are not enough points for integration.
- Integrate the integrand along radius using GSL.
- Subtract the total area blocked by the planet.
- Adjust the result if the planet is transparent.
- Normalize to the stellar radius.
- Return the modulation.

6.14.6 modulationm1:

Walkthrough:
- If toomuch was not reached, return -1.
- Find the impact parameter before and after tau reached toomuch.
- Use linear interpolation to calculate planet radius.
- Calculate and return the modulation assuming the planet is an opaque disc (R^2/R^2_*).

6.14.7 modulationperwn:

Variables Modified:
- Set `tr.modlevel` from `th.modlevel`.

Walkthrough:
- Read the modlevel flag to calculate the modulation using the optical-depth per impact parameter (modulation1) or an opaque disk of radius r = r(tau=toomuch).
- Call to `modulation1` or `modulationm1` depending on modlevel.
- Return the value given by `modulation1` or `modulationm1`.

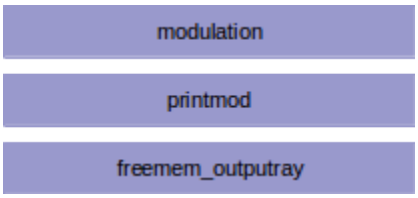

Figure 13: Function structure of observable.c. The boxes contain function names, and arrows point from a function to the function it calls. Functions are called from left to right, then top to bottom. Boxes are color-coded as follows: purple functions are used for eclipse geometry, blue functions are used for transit geometry, and green functions are used in both.

6.15 observable.c: EDITED

This file contains `modulation`, a routine which uses `modulationperwn` to calculate modulation at each wavenumber. Figure 13 shows the function structure of observable.c.

6.15.1 List of Functions Defined in observable.c:

`int modulation(struct transit *tr)`
Calculate the transit modulation at each wavenumber.

`void printmod(struct transit *tr)`
Print (to file or stdout) the modulation as function of wavelength.

`int freemem_outputray(struct outputray *out, long *pi)`
Free the transit modulation array.

6.15.2 modulation

Variables Modified
- Allocate `tr.ds.out.o` (modulation output) (FINDME: same issue as before).
- Call to `setgeom` from geometry.c to calculate `tr.ds.sg.x, tr.ds.sg.y` (coordinates of the center of the planet with respect to the star).
- Call to `modulationperwn` to calculate `tr.ds.out.o` (modulation).

Walkthrough
- Allocate modulation output.
- Check that tau, makeipsample, and makewnsample functions have been called.
- Call `setgeom` to calculate X and Y values (center of the planet with respect to the star). Note that these values are not currently used by the function, and are intended to be used to account for limb-darkening.
- Call `moldulationperwn` as sol.spectrum from slantpath.c to calculate modulation.
- Update the progress indicator to account for `TRPI_MODULATION`.
- Call printmod to print the modulation to file.

6.15.3 FINDME:

- `sg.x and sg.y` are not used. (also, `x, y` equations look fishy).

7 These Sections do not belong to the Code Doc!

7.1 Extinction Coefficient:

The molecular extinction coefficient is calculated following Equations (3.36)–(3.37) in cgs-Gaussian units as:

$$e_m = \frac{\pi e^2}{c^2 m_e} \sum_i \frac{\rho_i}{m_i} \frac{gf_i}{Z_i} \exp\left(-\frac{hcE^i_{\rm low}}{kT}\right) \left(1 - \exp\left(\frac{hc\bar{\nu}^i_0}{kT}\right)\right) \Psi(\bar{\nu}, \alpha_D, \alpha_L), \tag{1}$$

where gf_i is the weighted oscillator strength (`ltgf = tr.ds.li.lt.gf`), Z_i is the partition function (`ziso = tr.ds.iso.isov.z`), $\bar{\nu}^i_0$ is the line wavenumber (`wavn = 1/tr.ds.li.lt.wl`), $E^i_{\rm low}$ the lower state energy level (`ltelow = tr.ds.li.lt.elow`, in cm^{-1}), T is the atmospheric temperature (`temp = tr.atm.t`), ρ_i is the isotopic density (`densiso = tr.ds.iso.isov.d`), and m_i is the isotope's mass (`mass = tr.ds.iso.isov.m`). Ψ is the Voigt line profile (`profwn = profile`, in cm), where α_D, and α_L are the Doppler and Lorentz line-broadening widths. The constants in the equation are: e is the electron charge (in statC), m_e the electron mass, k the Boltzmann constant, h the Planck constant, c the speed of light.

The Doppler and Lorentz widths:

$$\alpha_D = \underbrace{\frac{\sqrt{2kT\ln 2}}{c}}_{\text{propto\_adop}} \frac{\bar{\nu}_0}{\sqrt{m_i}} \tag{2}$$

$$\alpha_L = \underbrace{\sqrt{\frac{2kT}{\pi^3 c^2}}}_{\text{propto\_alor}} \sigma_c \sum_{\rm coll} \frac{\rho_j}{m_j} \sqrt{\left(\frac{1}{m_j} + \frac{1}{m_i}\right)} + \underbrace{\alpha_N}_{\text{ignored}} \tag{3}$$

where σ_c is the cross section of the isotope (`csiso = tr.ds.iso.isov.c`), m_j is the mass of the colliding isotope, ρ_j is the density of the colliding isotope, and α_N is the natural broadening (which is negligible compared to the collisional broadening).

Note that, actually, α_D is the Doppler half-width at half maximum (where the Doppler width is defined as HWHM/$\sqrt{\ln 2}$). Since the Doppler profile depends on the wavenumber, the profile must be recalculated every certain range in wavenumber.

7.2 HITRAN Line Strength to gf Conversion:

The HITRAN database (**?**) provides the line intensity (cm^{-1}/(molecule cm^{-2})), which needs to be converted to the dimensionless gf value. Equation (20) of **?** gives the line intensity (S) in terms of the Einstein A_{21} coefficient:

$$S = A_{21} \frac{I_a g_2}{8\pi c \nu_0^2} \frac{1}{Z(T_0)} \exp\left(\frac{-hcE_1}{kT_0}\right) \left(1 - \exp\left(\frac{-hc\nu_0}{kT_0}\right)\right), \tag{4}$$

where I_a is the isotopic abundance, g_i the statistical weight of the level i, c the speed of light, ν_0 is the line wavenumber, Z is the partition function, T_0 is the HITRAN standard temperature, h the Planc constant, E_1 the lower state energy level (in cm^{-1}), and k the Boltzmann constant.

Replacing the Einstein A_{21} coefficient by the oscillator strength (f_{12}) from her Equation (36), where $epsilon_0$ is replaced by $1/(4\pi)$ when working in cgs units:

$$A_{21} = \frac{g_1}{g_2}\frac{8\pi^2 e^2 \nu_0^2}{m_e c} f_{12}, \tag{5}$$

where e is the electron charge (in statCoulomb), m_e the electron charge, and f_{12} the oscillator strength. Then, we have for the line intensity:

$$S = I_a \frac{g_1 f_{12}}{Z(T_0)}\frac{\pi e^2}{m_e c^2}\exp\left(\frac{-hcE_1}{kT_0}\right)\left(1-\exp\left(\frac{-hc\nu_0}{kT_0}\right)\right), \tag{6}$$

and finally:

$$gf = g_1 f_{12} = S\frac{m_e c^2}{\pi e^2}\frac{Z(T_0)}{I_a}\exp\left(\frac{hcE_1}{kT_0}\right)\left(1-\exp\left(\frac{-hc\nu_0}{kT_0}\right)\right)^{-1}. \tag{7}$$

7.3 Density Calculation:

The abundances in the atmosphere file can be given either as a mass fraction (μ, mass mixing ratio) or as a number fraction (ν, mixing ratio):

$$\mu_i = \frac{m_i n_i}{m}; \qquad \nu_i = \frac{n_i}{n}, \tag{8}$$

with n_i and m_i the mass and number of molecules of species i, m and n the total mass and number of molecules in the layer.

The density profiles are calculated using the ideal gas law in the form:

$$p = \frac{n}{V}kT, \tag{9}$$

with p, T, V the total pressure, temperature, and volume of the layer. Recognizing that the partial density can be written as:

$$\rho_i = \frac{m_i n_i}{V}, \tag{10}$$

and replacing the volume from Equation (9), we have:

$$\rho_i = \frac{m_i n_i}{n}\frac{P}{kT}, \tag{11}$$

wich can be restated as:

$$\rho_i = m_i \frac{n_i}{n}\frac{P}{kT} = m_i \nu_i \frac{P}{kT}, \tag{12}$$

or as:

$$\rho_i = \frac{\mu_i m}{n}\frac{P}{kT} = \bar{m}\mu_i \frac{P}{kT}, \tag{13}$$

with $\bar{m} = m/n$, the mean molecular mass in the layer.

7.4 Radiative Transfer:

7.4.1 Some definitions to begin:

The specific intensity (I_ν) is the energy ($\mathrm{d}E$) between frequencies ν and $\nu + \mathrm{d}\nu$ per unit time ($\mathrm{d}t$) that flows through a unit surface area ($\mathrm{d}A$) at an angle θ (measured with respect to the normal) contained in a solid angle ($\mathrm{d}\Omega$):

$$I_\nu = \frac{\mathrm{d}E}{\cos\theta \mathrm{d}A \mathrm{d}\Omega \mathrm{d}\nu \mathrm{d}t}, \tag{14}$$

with units of ergs $\mathrm{s}^{-1}\mathrm{cm}^{-2}\mathrm{sr}^{-1}\mathrm{Hz}^{-1}$. The specific flux ($F_\nu$) is the net energy between frequencies ν and $\nu + \mathrm{d}\nu$ per unit time that flows perpendicularly through a unit surface area, i.e., the specific intensity integrated over solid angle:

$$F_\nu = \int I_\nu cos\theta \mathrm{d}\Omega \tag{15}$$

with units of ergs $\mathrm{s}^{-1}\mathrm{cm}^{-2}\mathrm{Hz}^{-1}$.

7.4.2 On to the radiative transfer equation:

When a ray of light passes through a medium its intensity decreases due to the absorption or scattering from particles. This is proportional to the intensity itself, to the medium density (ρ), to the absorption coefficient (or opacity, κ_ν) and the path traveled ($\mathrm{d}s$):

$$\mathrm{d}I_\nu = -I_\nu \kappa_\nu \rho \mathrm{d}s. \tag{16}$$

The opacity is the cross section of photons at frequency ν per unit mass. Additionally, the medium can also contribute to the intensity, this is described by the emission coefficient (j_ν), the rate of change in intensity by the emission coefficient is proportional to the density and the path traveled:

$$\mathrm{d}I_\nu = j_\nu \rho \mathrm{d}s. \tag{17}$$

Let's define the source function $S_\nu = j_\nu/\kappa_\nu$ and the optical depth (χ) as $\mathrm{d}\chi = \kappa_\nu \rho \mathrm{d}s$, the transfer equation becomes:

$$\frac{\mathrm{d}I_\nu}{\mathrm{d}\chi} = -I_\nu + S_\nu. \tag{18}$$

7.4.3 Emergent Flux:

To study the case of the flux emitted by a planet (or star), first assume the plane-parallel approximation, which is valid when the vertical scale is much smaller than the horizontal scale. In this case we can safely assume that the atmosphere is composed by a stratified set of plane slabs with uniform properties.

Defining $\tau = -\kappa_\nu \rho \mathrm{d}z$ as the vertical optical depth (note the negative sign in the definition, that implies that $\tau = 0$ at the top of the atmosphere, increasing inward). The path for a ray ($\mathrm{d}s$) with an angle θ with respect to the vertical is related to the vertical path as: $\mathrm{d}s = \mathrm{d}z/\cos\theta \equiv \mathrm{d}z/\mu$, and thus the RT equation becomes:

$$-\mu \frac{\mathrm{d}I_\nu}{\mathrm{d}\tau} = -I_\nu + S_\nu. \tag{19}$$

To sove, multiply by $\exp(-\tau/\mu)$:

$$-\mu\frac{\mathrm{d}I_\nu}{\mathrm{d}\tau}e^{-\tau/\mu} + I_\nu e^{-\tau/\mu} = S_\nu e^{-\tau/\mu} \quad (20)$$

$$-\mu\frac{\mathrm{d}}{\mathrm{d}\tau}\left(I_\nu e^{-\tau/\mu}\right) = S_\nu e^{-\tau/\mu} \quad (21)$$

For the emerging intensity at the top of the atmosphere ($\tau = 0$), consider a depth where the atmosphere is well optically thick, $\tau = \tau_b$, such $exp{-}\tau_b/\mu \to 0$, then:

$$-I_\nu e^{-\tau/\mu}\Big|_0^{\tau_b} = \int_0^{\tau_b} S_\nu e^{-\tau/\mu}\mathrm{d}\tau/\mu \quad (22)$$

$$I(0) = \int_0^{\tau_b} S_\nu e^{-\tau/\mu}\mathrm{d}\tau/\mu \quad (23)$$

Under LTE, the source function becomes the Planck function $B_\nu(T)$. To obtain the emergent flux, integrate the solid angle over the half sphere:

$$F_\nu = \int_0^{2\pi}\int_0^{\pi/2} I_\nu \cos\theta \sin\theta \mathrm{d}\theta\mathrm{d}\phi = 2\pi\int_0^{\pi/2} I_\nu \cos\theta\sin\theta\mathrm{d}\theta = 2\pi\int_0^1 I_\nu\mu\mathrm{d}\mu. \quad (24)$$

Combined with Equation 23:

$$F_\nu(0) = 2\pi\int_0^1\int_0^{\tau_b} B_\nu e^{-\tau/\mu}\mathrm{d}(\tau/\mu)\,\mu\mathrm{d}\mu \quad (25)$$

To solve this equation, use the diffuse approximation where we replace $\tau/\mu \approx \tau/\bar{\mu}$, with $1/\bar{\mu} = 5/3 = 1.66$, the diffusivity factor. Then:

$$F_\nu(0) = 2\pi\int_0^{\tau_b} B_\nu e^{-\tau/\bar{\mu}}\mathrm{d}(\tau/\bar{\mu})\int_0^1 \mu\mathrm{d}\mu \quad (26)$$

$$= 2\pi\int_0^{\tau_b} B_\nu e^{-\tau/\bar{\mu}}\mathrm{d}(\tau/\bar{\mu})\,\frac{1}{2} \quad (27)$$

Finally the emerging flux at the top of the atmosphere is:

$$F_\nu(0) = \pi\int_0^{\tau_b} B_\nu e^{-\tau/\bar{\mu}}\mathrm{d}(\tau/\bar{\mu}) \quad (28)$$

8 Equations

Optical depth:

$$\tau = \int e\cdot\mathrm{d}s \quad (29)$$

where e is extinction and $\mathrm{d}s$ is the differential path element.

Extinction:

$$e = \quad (30)$$

where **(FINDME: (FINDME))**

Modulation:

$$M_\lambda = \frac{1}{R_\star^2}\left(R^2 - 2\int_0^R \exp^{-\tau_\lambda(r)}\, r\, \mathrm{d}r\right) \tag{31}$$

where R_* is the stellar radius, τ_λ is optical depth at a particular wavelength as a function of radius, and R is the planetary radius.

Planck function for wavenumbers:

$$B_\nu = 2h\bar{\nu}^3c^2\frac{1}{\exp(\frac{h\bar{\nu}c}{k_BT}) - 1} \tag{32}$$

where ν is wavenumber, h is the Planck constant, c is the speed of light, k_B is the Boltzmann constant, and T is temperature.

Emergent intensity:

$$I = \int_0^{\tau_{max}} B_\nu \mathrm{e}^{-\tau}\mathrm{d}\tau \tag{33}$$

where B_ν is the Planck blackbody function and τ is optical depth. The integral is from 0 to `tr.toomuch`.

Flux:

$$F = \sum_{i=1} \pi I_i((\sin\theta_{fin})^2 - (\sin\theta_{in})^2) \tag{34}$$

where I is intensity, A is area, n is the number of angles, and w_n is the number of wavenumbers.

Hydrostatic pressure:

$$\frac{dP}{P} = -\frac{dz}{H} \tag{35}$$

where P is pressure, z is height, and H is scale height.